股票熱線—應用 LSTM 分析、預測股票數據之 LINE BOT

魏凱城(指導教授)/曾詮智/陳育騰/林承璉/洪莛瑀/鄭旭翔

國立彰化師範大學資訊工程學系

500 彰化市進德路一號

Email: kaicheng46@gmail.com

一、摘要

近年來學會如何投資理財已經變成大家必備的能力。如何在最短時間內，將自己的利益最大化成了一個艱難的問題。而我們想利用 LSTM 演算法搭配 line bot 來預測股票的走勢，並希望以簡單、明瞭的操作介面讓各個年齡層的人都能輕鬆使用。若是能準確預測，那便可在投資方面占得先機、提前部署，減少不必要的虧損。

Summary

In recent years, learning how to invest in financial management has become an essential ability for everyone. We want to use the LSTM algorithm with line bot to predict the trend of the stock, and hope that with a simple and clear operation interface, people of all ages can easily use. If it can be accurately predicted, it can take the lead in investment and deploy in advance to reduce unnecessary losses.

二、研究目標

鑒於近年來銀行存款低，投資房地產又需要一筆不小的資金，股票自然就成為了投資客們心目中的首選。股票入門容易但達到期望的投資目標卻不容易，在網路及媒體上的五花八門的投資專家，如果自己想要自己研究，光是均線、KD 指標、MACD 指標…等不同技術分析就已經夠讓人焦頭爛額了，而且在各領域的股票價格隨時會受到不同方面的影響，因此建立一個能在不同領域都能適用的模型並確保能不在瞬息萬變的股市中成為刀下亡魂成為了我們的首要目標，讓除了股票新手可以加以使用外，經驗老到的投資客也能藉由模型選出合適的策略

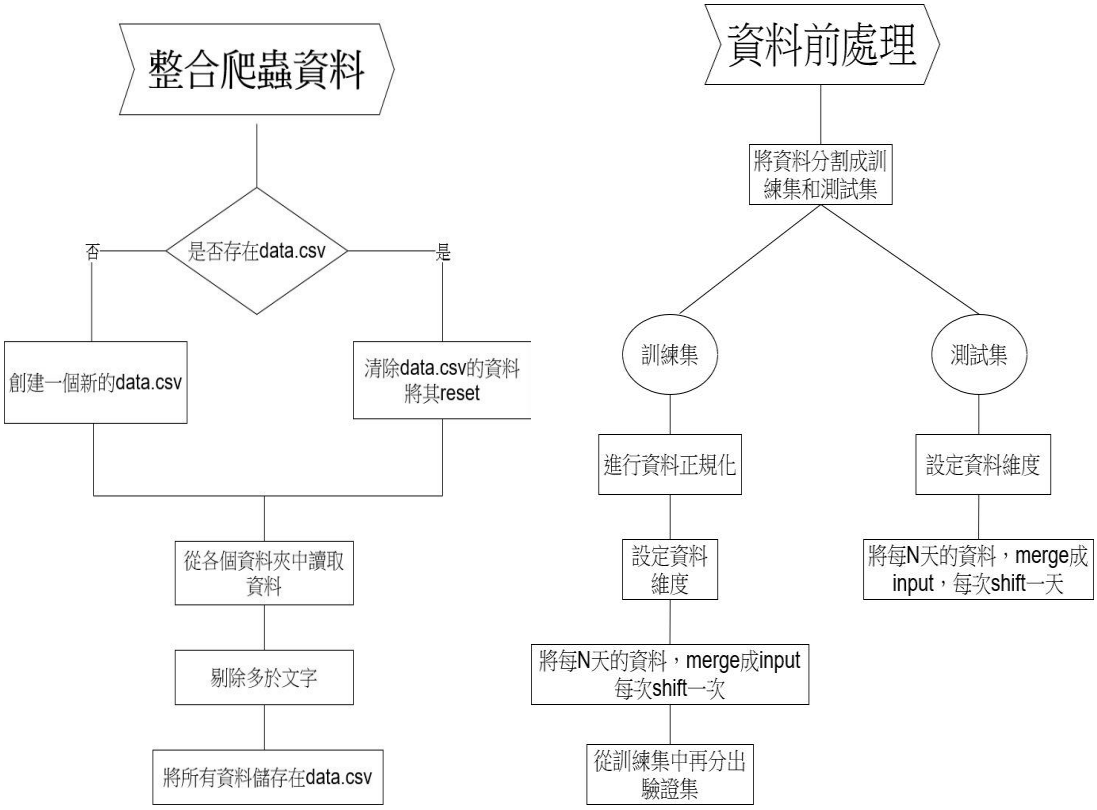
目前的投資市場上有非常多的股票供以選擇，配合機器學習不同模型的應用進行技術分析，能準確預測短期內股市的變動，並搭配 LINE BOT 作為輸出介面供使用者隨時隨地查詢，以下是我們的研究目標：

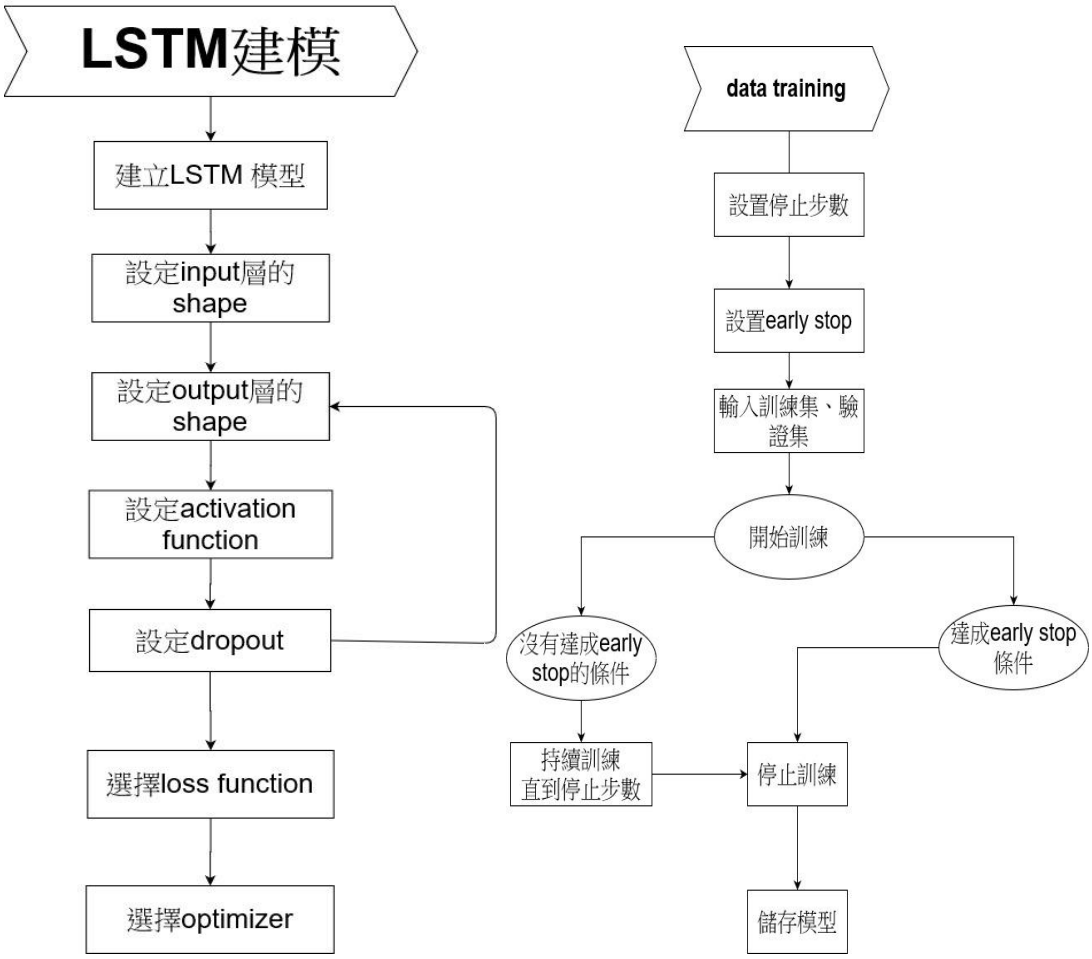
1. 挑選出不同類別的股票進行預測。
2. 運用機器學習模型進行訓練及修正。
3. 將預測結果與實際結果進行比較。
4. 將成果置入 LINE BOT。

三、整體架構

為了完成一個能夠提供預測股票資料服務的 LINE BOT，我們需要準備以下的東西:
歷年的個股日成交資訊、經過訓練的 LSTM 模型(每個股票一個)、LINE BOT 主程式、每日更新資料之程式

首先，我們需要每股都各訓練出一個模型，程式流程圖如下:





接著，我們會有一個用 python 寫的 LINE BOT 的主程式，在伺服器上 24 小時不間斷地運作，在開機的時候載入 models 資料夾內的.h5 模型檔案，並儲存在 dictionary 內，隨時對使用者下的指令做 response，而可用的指令如附圖

而以下就重點的股票功能做解釋:

1.股票列表:

我們將有訓練好模型的股票之名字以及序號以 json 檔的格式存在 stock.json 這個檔案內，格式如以下範例:

```
{
    "黑松": [1234, "食品工業"],
    "台塑": [1301, "塑膠工業"]
}
```

而此指令便是將有提供服務的股票展示出來，讓使用者一目了然

2.股價查詢 序號 or 名字:

此功能可以讓使用者查詢到近十筆的股票資料，

可以展示日期、成交股數、成交金額、開盤價

、最高價、最低價、收盤價、漲跌價差、

總成交筆數

3.預測股價 序號 or 名字:

機器人在剛開機的時候就會載入所有的模型，將其存在 model 這個 dictionary 中，當接收到這個指令的時候，程式就會調用之前讀進來的模型，再調用 predict.py 內的函式進行預測的動作

4.模型展示 序號 or 名字:

此功能可以讓使用者查看有關訓練好的模型的各種參數，透過 model.summary(print_fn=lambda x: stringlist.append(x))這個函式，將訓練好的模型

型架構展示給使用者看

5.測試結果 序號 or 名字:

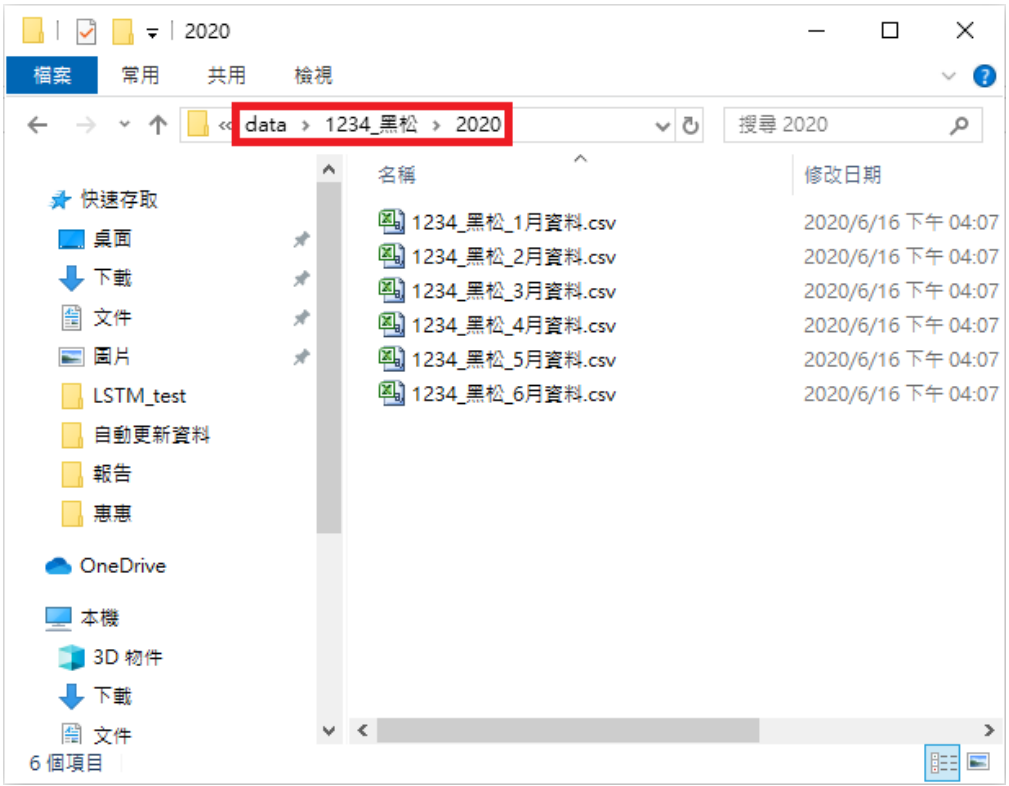
此功能會從 stock_images 資料夾內貼出模型 testing 產生的曲線圖，將預測準確度以可視化的圖形展示出來

除了這個之外，還會有一個小程式每日更新股票資訊，平時保持 sleep 的狀態，不會耗用內存，在每天的 23:59，他會結束 sleep 開始跑迴圈，直到 00:00 一到，就會抓取 stock.json 內紀錄的股票名字以及序號，並將其傳入函式，更新前一天的個股日交易資訊

四、開發過程

- 爬蟲程式:

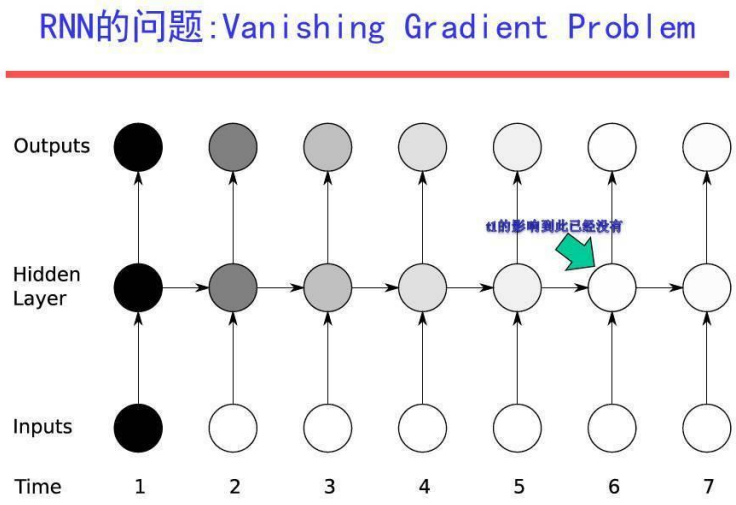
  訓練 LSTM 模型需要大量的資料，而台灣證券交易所的公開網站可以查到長達十年的個股日成交資訊，因此，我們寫了一個爬蟲程式將這些資料逐一抓取下來，使用 pandas 套件的 read_csv 函式，從台灣證券交易所個股日成交資訊頁面抓取 csv 檔案，並將他們分類存放在”data/序號_名稱/年份”資料夾內，如附圖:

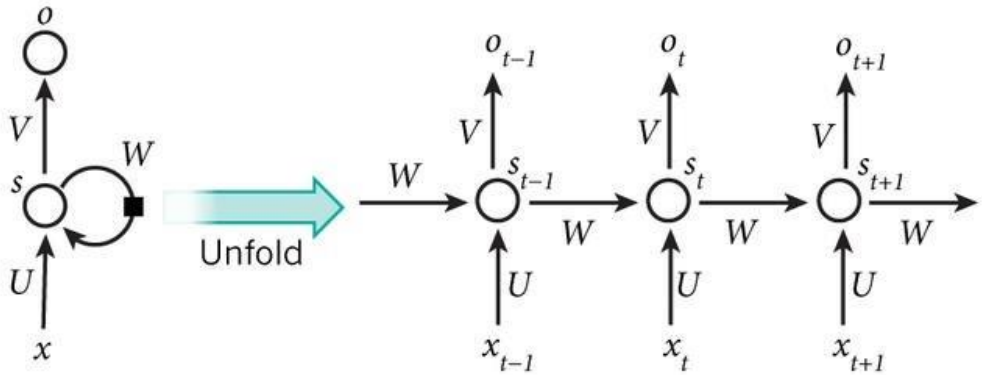


原本是要使用 request 套件來進行爬蟲的工作，可以使用 get、post 等各種函式來快速抓取資料,還可以自己寫 User-Agent 填入 headers 的參數來偽裝成使用者的瀏覽器向伺服器發出 request。但是後來實作發現被伺服器 block 的主因是因為短時間內一次性爬取太大量的資料，被伺服器誤認為 DDOS，因此在程式內添加 time.sleep()，增加抓取每個 csv 檔案的間隔時間，就可以簡單的解決這個問題，因此後來便改用 pandas 的 read_csv，並直接填入 url 就可以抓取資料了

- 神經網路:

1.RNN: (圖片來源: https://kknews.cc/news/6jnmq3.html)





Xt：是時間 t 處時的輸入。

St: 是時間 t 處時的”記憶”，St = f(UXt + WSt-1)，函數 f 通常是非線性的，

例如:tanh、Relu,或 Sigmoid...等。

但通常不能獲取之前過長時刻的信息。

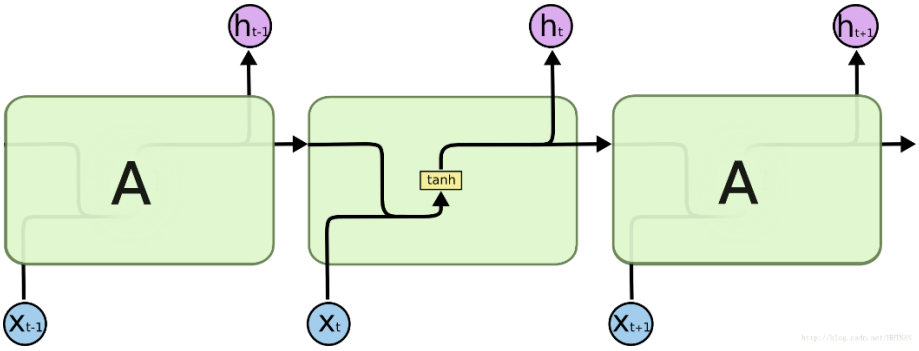
Ot: 是時間 t 處時的輸出。例如,如果我們想預測一個句子的下一個詞，

它將會是一個詞彙表中的機率向量，Ot = softmax(V*St)。

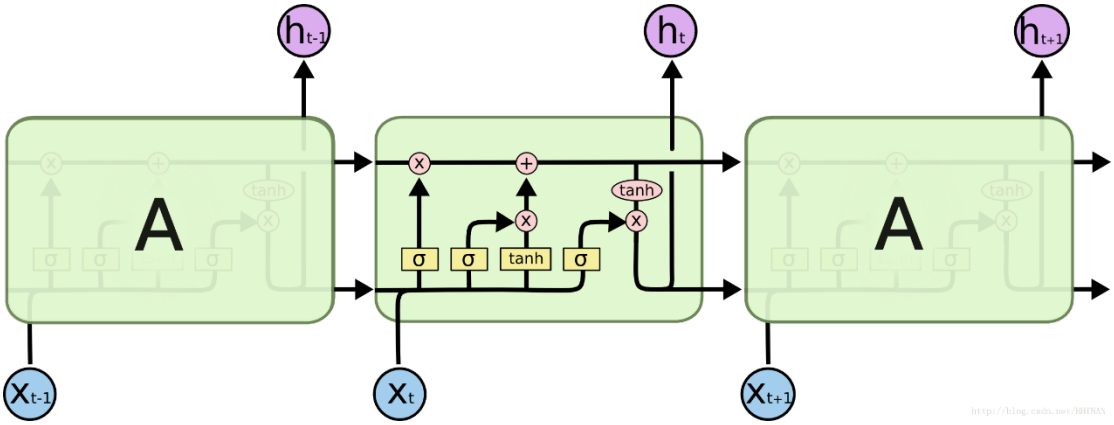
不像傳統的神經網絡，在不同的 layers 裡會使用不同的參數, RNN 在所有步驟中都共享參數

（U、V、W）。這表示我們在每個步驟上執行的任務都相同，僅僅是輸入不同而已。這個機制大大的減少了我們需要學習的參數的數量。

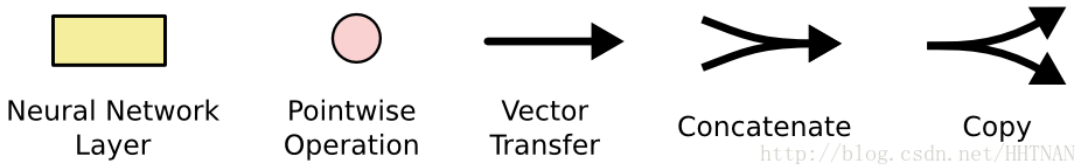
2.LSTM: (圖片來源: https://www.itread01.com/content/1549269013.html)



A. 傳統的 RNN 在每一步的隱藏單元裡,只是執行一個簡單的激活函數（Activation functions）操作，例如:tanh, Relu



B. LSTM 裡每個迴圈的模組裡會有 4 層結構: 3 個 sigmoid 層，1 個 tanh 層

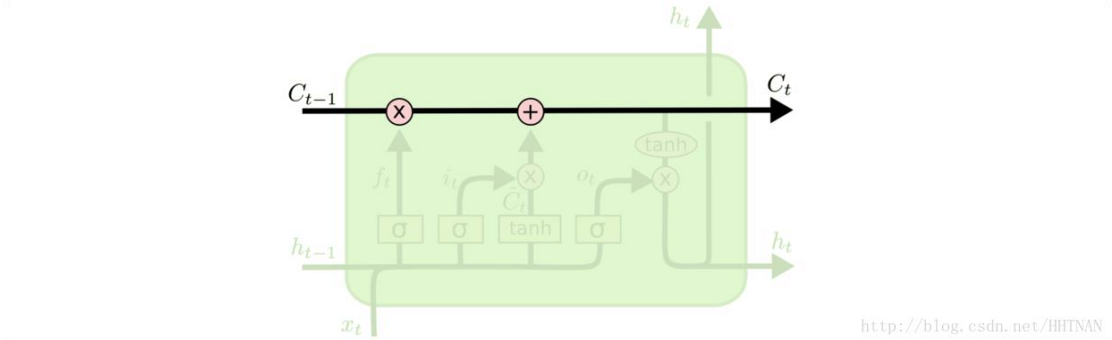


a. Neural Network Layer: 在模組裡的神經網絡層，可以看作一個個的非線性處理模塊。
b. Pointwise Operation: 逐點運算，舉個逐點乘法的例子就是:0.5×[1, 2, 3] = [0.5, 1.0, 1.5]。
c. Vector Transfer: 一條直線，信息傳遞方向。
d. Concatenate: 兩條直線交會，表示信息的匯合。
e. Copy: 一條直線分為二，與上面對應，表示信息的複製。

C.LSTM 内部結構詳細介紹:

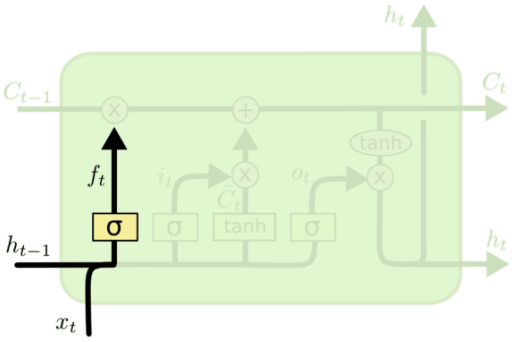
1. 細胞狀態的傳送帶:

一條水平傳送帶，這上面只有少量的線性操作，傳送的時候會發生一些信息的交互，信息就在這上面一直保存。可以在這條傳送帶上取值，也可以在上面輸入值。 Ct 是 memory，也就是記憶。



2. 決定遺忘多少信息:

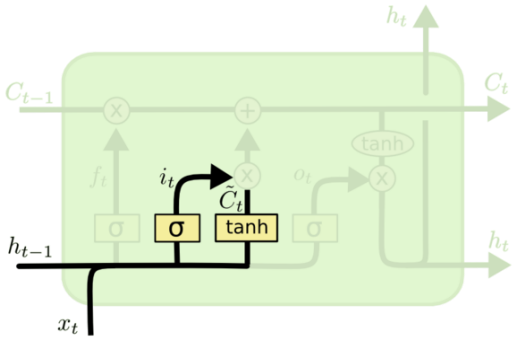
第一層是個遺忘層，用於決定細胞狀態中需要遺忘什麼信息。把 ht−1 和 xt 拼接起來，傳給一個 Sigmoid 函式，來一起去決定要以多大的程度來忘記這個內容，而這個函式會輸出 0 到 1 之間的值，這個值乘到細胞狀態 Ct−1 上去。Sigmoid 函式的輸出值直接決定了狀態資訊要保留多少。換言之，就是 ft 要以多大的機率留下這個信息，這個式子產出一個概率值。W 和 b 都是參數，會由訓練得到。



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

3. 確認更新的信息:

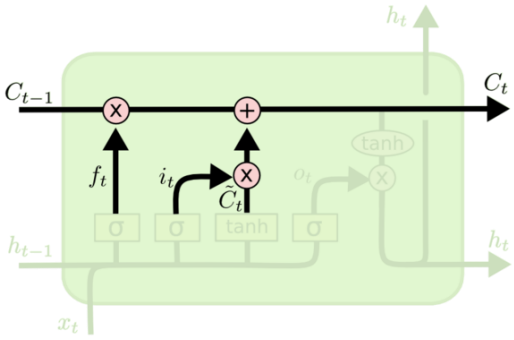
由於上個步驟用遺忘 gate 把部份信息遺忘掉，就應該再補上一部分新的信息過來到 Cell State 中，也就是在步驟一裡所說過的，可在傳送帶上輸入值。這個步驟裡的兩個值: it 和$\tilde{c}t$，前者是概率，後者是記憶; it 是一個 0~1 的概率值, it 的作用就是對目前爲止學到的所有信息做一個過濾，由這個的概率值可以看出，現在學到的哪部分新知識可以更新之前所記憶的。本次學到的知識就是$\tilde{c}t$，要將它加入之前學到的所有信息中，而 tanh 的輸出在-1~1 之間，說明 cell 狀態在某些維度上需要加強，在某些維度上需要減弱



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

4. 更新 cell 狀態:

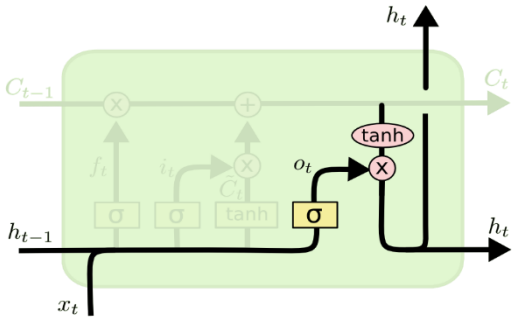
讓舊的 cell 狀態 Ct−1 與 ft (f 是 Forget 的意思)相乘來遺忘部分信息，然後再加上需要更新的部分 it∗$c^{-}t$ (i 是 Input 的意思)，最後生成了新的 cell 狀態 Ct。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

5. 輸出信息:

Ot 表示，先運行一個 Sigmoid 層來確定 cell 狀態將有多少信息輸出出去。再將新的 cell 狀態(Ct) 通過 tanh 進行處理(得到-1~1 之間的值)，並將其與 Ot 相乘，產生 ht，表示確定要輸出的 cell 狀態。



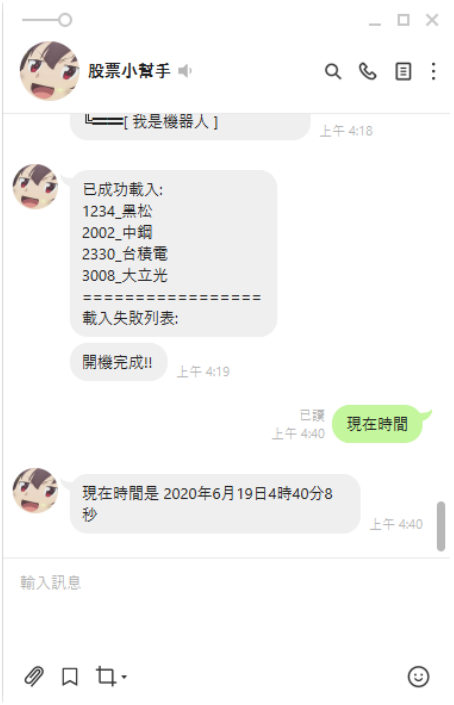
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- LINE BOT 主程式:

  在選擇人機介面時，我們挑選了 LINE 作為我們的介面，原因如下:

  (1)LINE 的使用率高，現在社群軟體興盛，幾乎大家都擁有 LINE 帳戶，除了能夠擁有高普及率之外，使用者也不必再額外安裝程式、或是申請帳戶，只要將機器人加入好友即可立刻開始使用

  (2)負責此項目的組員曾經開發過 LINE BOT 程式，有相關經驗，能以較少人力完成更多項目



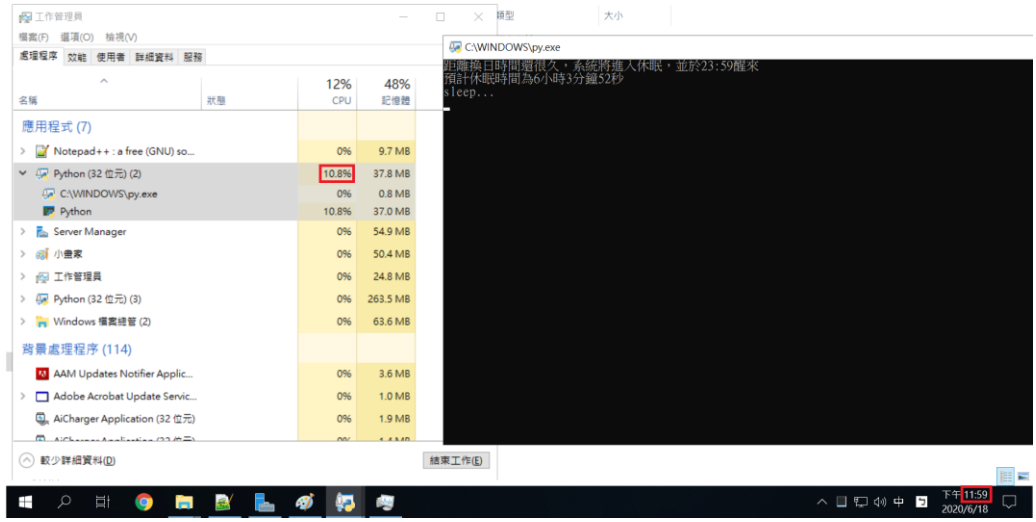
  LINE 機器人的運作原理很簡單，程式會偽裝成使用者的裝置登入 LINE，當接收到伺服器的封包後，從其內容的各種參數來傳送不同的 signal 給主程式，接著，主程式就可以依不同 signal 做出不同的反應(Ex. op.type 為 26 為機器人接收到其他用戶傳來的訊息，此時便可以設定接到何種文字指令會做什麼反應)

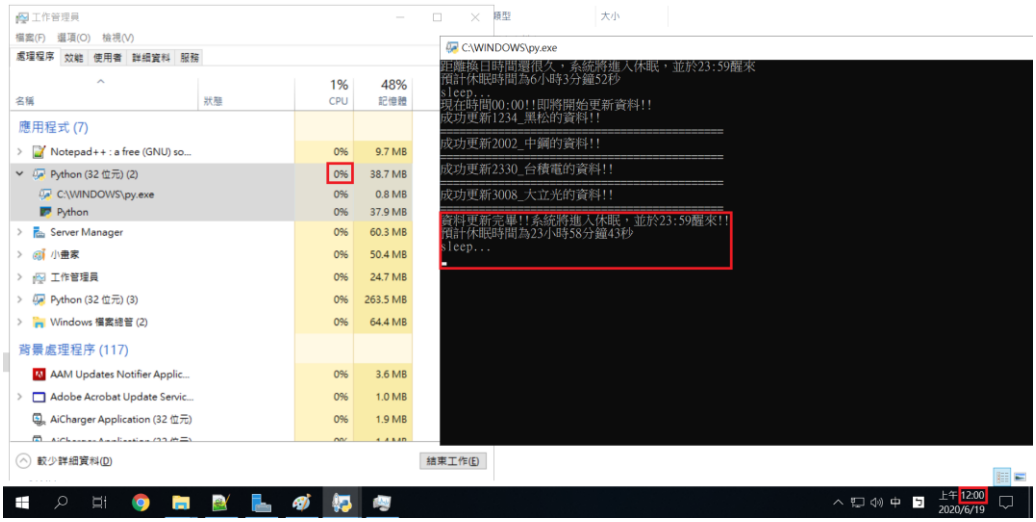
  不過，這個機器人程式並非常見的那種 LINE 官方帳號之機器人，他更像是透過非官方途徑達成這個目標的，因此也自然不是使用官方 API，而是使用一位國外作者寫的”linepy”這個 API(來源: https://github.com/fadhiilrachman/line-py)，而使用這非官方途徑自然是有利也有弊。官方的機器人的免費方案的機器人帳號限制多，只能做到接話回話功能，無法自行查看帳戶收到甚麼訊息，也無法主動傳訊息給其他用戶。相較之下，非官方途徑因為是使用特殊方式登入帳戶，並非像官方那樣是直接用官方帳號，因此可以自由查看該帳號收到的訊息，程式上設計也會比較自由，不會有限制。不過，原本的 API 直接照用的話是無法使用的，因為 LINE 官方那邊會不定期更新版本限制，而每更新一次往往都會使不少版本過舊的非官方機器人陣亡，不過在 source code 內修改一些版本相關的 code 就可以解決這個問題了，加上負責此項目的組員的開發經驗也都是用非官方途徑，因此最後我們決定以非官方的方式來開發 LINE BOT。

- 每日更新資料之程式:

  其實原本我們是想要將這個功能放在機器人的程式內，讓他一換日就暫停服務去更新資料，可是測試過程我們發現了一個問題，要是沒有訊息進來，機器人便會保持休眠的狀態。因此，我們只好額外寫一個程式，平時讓它休眠，在換日前便會醒來，並準時在 00:00 時，用前面使用過的爬蟲技巧來下載最新的資料，這樣就可以確保我們的資料永遠都是最新的

  - 執行畫面以及休眠與否的 CPU 使用量比較:





五、遭遇的困難及問題

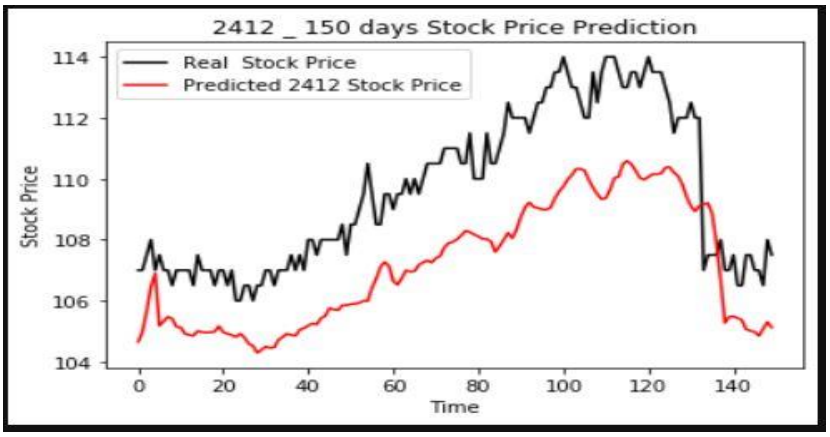
- 爬蟲時被伺服器 BLOCK:

  一開始遇到這個問題的時候，我們上網找了不少資料，使用了 User-Agent 並將其填入 headers 參數，接著再加入 time.sleep()，慢慢調整延遲時間，希望能用最少的時間趴到最多的資料同時不會被視為 DDOS 阻擋，而後我們發現用 read_csv 就可以抓了，只要 sleep 時間掌控的 OK，也不會被阻擋下來，因此便改成了現在的這個樣子

- 如何將模型載入到機器人內:

  為了將訓練好的模型載入程式內，我們思考過好幾種儲存方式，例如為了節省空間，輸入指令才載入模型，不過後來還是決定直接在程式剛執行的時候就花久一點的時間把全部的模型載入進來，否則輸入指令再載入會花費大量時間，導致使用者感覺延遲太久，效率太差

- 預測結果與實際結果有明顯上下位移:



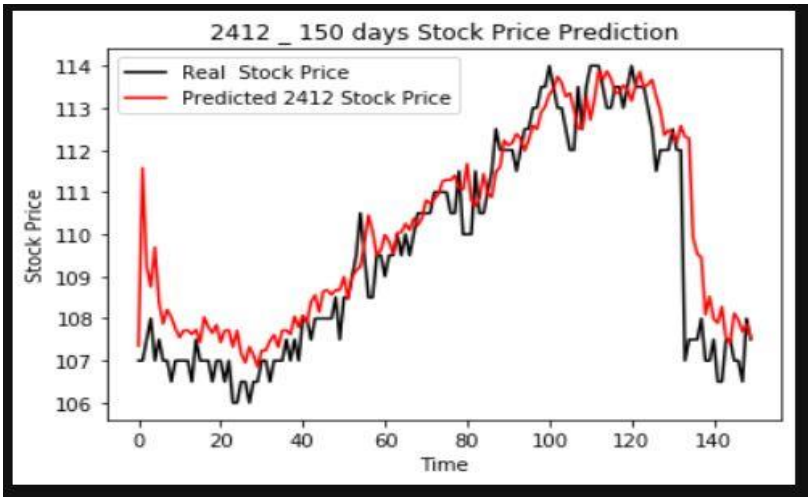
推測原因:

每層 layer 中的節點個數太少，無法紀錄過多的特徵，產生有上下的誤差

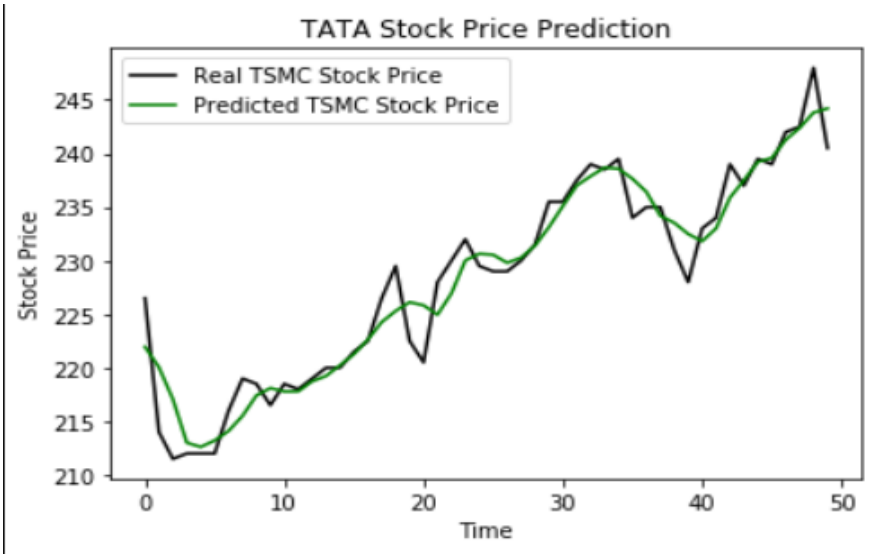
解決方法:

增加每層中 unit 的數量

增加 hidden 層



- 遇到過擬合的現象:



推測原因:

訓練數據太少，導致預測結果不如預期。

解決方法:

增加訓練集的資料量

加上 dropout

- 每種股票適合的 LSTM 模型都不同

  推測原因:

  依據使用的 activition function 不同，效果都會落差。

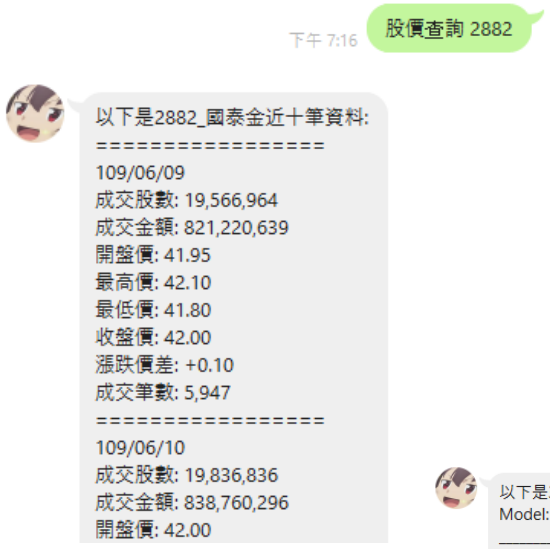
- 解決方法:

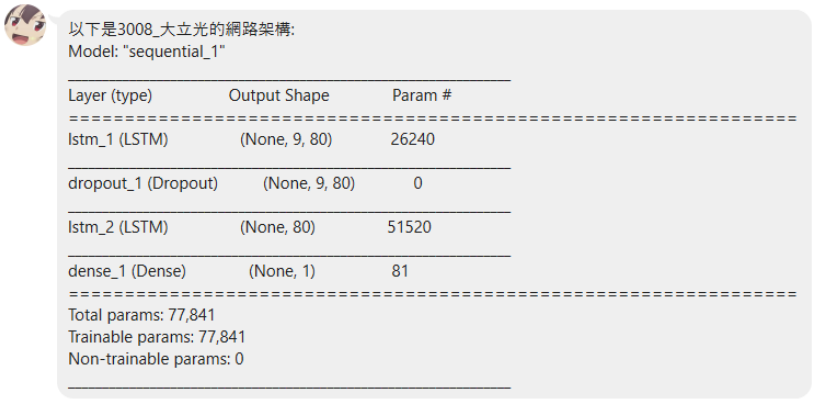
  我們手動測試了 15 支的股票，並做了準確率和相關係數的計算，在同樣的模型、普遍情況下，使用 relu activation function 的準確率和相關係數皆大於 tanh activation function，而 hidden layer 的層數在兩層到三層間為佳

六、成果

將機器人加入好友(LINE ID: mymeguminbot)便可收到歡迎訊息，照著歡迎訊息的指示輸入”help”指令便可看到指令清單

以下是股票相關功能的展示:





股票代碼	1234_黑松					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.99295	0.98889	0.99264	0.99135	0.97556	0.94022
相關係數	0.94148	0.94556	0.93722	0.94223	0.93985	0.98756
股票代碼	2317_鴻海					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.95895	0.97950	0.96018	0.98004	0.98099	0.97733
相關係數	0.95780	0.95781	0.94981	0.95614	0.95498	0.94977
股票代碼	2002_中鋼					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.98826	0.99152	0.98089	0.99161	0.97594	0.98035

相關係數	0.98389	0.98358	0.98211	0.98382	0.98378	0.98328
股票代碼	3008_大立光					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.96112	0.95974	0.97353	0.94492	0.97352	0.96786
相關係數	0.94499	0.94495	0.94448	0.94471	0.94538	0.94445
股票代碼	2454_聯發科					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.97683	0.95484	0.97398	0.97613	0.97331	0.97249
相關係數	0.98184	0.98108	0.97928	0.98180	0.98176	0.98187
股票代碼	1301_台塑					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.98422	0.98290	0.98027	0.98393	0.98236	0.98217
相關係數	0.97706	0.97371	0.97584	0.97625	0.97735	0.97679
股票代碼	2330_台積電					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.97710	0.97551	0.90883	0.97613	0.93183	0.91932
相關係數	0.98186	0.84536	0.93055	0.98157	0.97782	0.88954
股票代碼	1303_南亞					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.98531	0.97178	0.98420	0.98627	0.98591	0.98077
相關係數	0.97399	0.97243	0.9754	0.97370	0.97431	0.97323
股票代碼	2412_中華電					
激勵函數	relu			tanh		
層數	2	3	4	2	3	4
準確率	0.99465	0.97667	0.99261	0.99440	0.99344	0.99004
相關係數	0.94444	0.91707	0.91437	0.94399	0.94429	0.94416
平均準確率	0.97993	0.97571	0.97190	0.98053	0.97476	0.96784
平均相關係數	0.96526	0.94684	0.95434	0.96491	0.96439	0.95896