



資料分析與學習基石：HW1_Kaggle

Dataset Analysis

Category	Course
Course	資料分析與學習基石
Tags	Dataset Kaggle
Last Edited Time	@March 8, 2022 9:08 PM
Done	<input checked="" type="checkbox"/>

HW1 : First visit in Kaggle data

Dataset

Description

Dataset Analysis

[Notebook #1 Recurrent Neural N. \(RNN\) Tutorial for Beginners](#)

[Notebook #2 Long Short-Term Memory N. \(LSTM\) for Beginners](#)

[Conclusion & Insight](#)

Reference

HW1 : First visit in Kaggle data

Dataset

在找 Dataset 時對下面幾個比較有興趣：

- [Huge Stock Market Dataset](#)：收集了2017以前美國證券交易所的 Trending 股票資訊。
- [YouTube Dislikes Dataset](#)：收集 USA、Canada、Great Britain 從2020年8月到2021年12月的 Youtube 發燒影片的資訊。

雖然 Youtube Dislike 感覺很有趣，但 Notebook 比較少可以研究，同時其他課也有跟 Stock 相關的課題，因此最後選擇 Huge Stock Market Dataset 做分析。

Description

- Title : Huge Stock Market Dataset
- Author : Boris Marjanovic
- Description : 由於大部分高品質的金融資料都需要花費金錢來取得，免費提供的資料集跟日本製造的壓縮機一樣稀少，因此作者描述此資料集是可以免費獲得的最好之一。

Dataset Analysis

整個資料集以CSV的格式儲存在 .txt 檔案裡(如下圖所示)：

```
Date,Open,High,Low,Close,Volume,OpenInt  
1999-11-18,30.713,33.754,27.002,29.702,66277506,0  
1999-11-19,28.986,29.027,26.872,27.257,16142920,0  
1999-11-22,27.886,29.702,27.044,29.702,6970266,0  
1999-11-23,28.688,29.446,27.002,27.002,6332082,0  
1999-11-24,27.083,28.309,27.002,27.717,5132147,0  
1999-11-26,27.594,28.012,27.509,27.807,1832635,0  
1999-11-29,27.676,28.65,27.38,28.432,4317826,0  
1999-11-30,28.35,28.986,27.634,28.48,4567146,0  
1999-12-01,28.48,29.324,28.273,28.986,3133746,0  
1999-12-02,29.532,30.375,29.155,29.786,3252997,0  
1999-12-03,30.336,30.842,29.909,30.039,3223074,0  
1999-12-06,30.547,31.348,30.505,30.883,2385046,0  
1999-12-07,30.883,31.052,29.909,30.547,2348161,0  
1999-12-08,30.547,30.795,30.249,30.505,2000481,0  
1999-12-09,30.547,31.012,30.547,30.924,2150096,0
```

Stock.a.us.txt 的前幾行內容

ETFs的資料集有 1344 個，而Stock的有 7195 個，總共有 8539 個檔案。

每一行都是某一天的股票資訊，按以下順序排列：

1. **Date**：日期。
2. **Open**：開盤價。
3. **High**：當日最高價。
4. **Low**：當日最低價。
5. **Close**：收盤價。
6. **Volume**：成交量。
7. **OpenInt**：持倉量。



持倉量是期貨市場獨有的概念，指某一時點上某契約未平倉契約的數量，類似於股票市場中的流通股本，只不過流通股本只有多頭。

成交量和持倉量為不同的指標，成交量通常代表著市場的迫切性；而持倉量增加，表明資金在流入市場，多空雙方對價格走勢分歧加大；持倉量減少則表明資金在流失，多空雙方的交易興趣下降。通常用來確認趨勢的強度。



證券交易通常有兩種操作，買方看漲行情做多頭，看跌行情做空頭。

開倉：無論是做多還是做空，下單買賣就稱為開倉。

平倉：多頭將買入的股票賣出、空頭買回賣出的股票，就稱為平倉。

透過分析每日個股成交量、漲跌情況，我們可以訓練模型來預測個股未來的漲跌情況，成為我們在股市投資的輔助指標之一，藉此省下不少技術面分析的時間。

基本上，這個 Dataset 和台灣證券交易所提供的個股資料是相同的，因此我認為可以學習這個 Dataset 以及其相關模型應用的實例，再延伸至台股、虛擬貨幣。

```

{
  stat: "OK",
  date: "20220306",
  title: "111年03月 2330 台積電          各日成交資訊",
  fields: [
    "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"
  ],
  data: [
    + [ ... ],
    + [ ... ],
    + [ ... ],
    + [ ... ],
    - [
      "111/03/07",
      "97,167,600",
      "56,147,025,127",
      "580.00",
      "581.00",
      "575.00",
      "576.00",
      "-19.00",
      "297,917"
    ]
  ],
  notes: [
    "符號說明:+/-/X表示漲/跌/不比價",
    "當日統計資訊含一般、零股、盤後定價、鉅額交易，不含拍賣、標購。",
    "ETF證券代號第六碼為K、M、S、C者，表示該ETF以外幣交易。"
  ]
}

```

台灣證券交易所上市個股日成交資訊，以3月6日的台積電(2330)為例。

Notebook #1_Recurrent Neural N. (RNN) Tutorial for Beginners

- Title : Recurrent Neural N. (RNN) Tutorial for Beginners
- Author : RAFET CAN KANDAR
- Upvoted : 62
- Medal : Silver
- Introduction : 此 Notebook 會先簡短介紹要使用的 model，並介紹RNN的應用範圍、為什麼使用 RNN。

▼ 步驟一：讀取資料

利用 `pandas` 讀取資料並顯示前幾列：

```
In [24]:  
data = pd.read_csv("/kaggle/input/price-volume-data-for-all-us-stocks-etfs/Stocks/ tsla.us.txt")
```

```
In [25]:  
#Let's examine a few examples from our data.  
data.head()
```

Out[25]:

	Date	Open	High	Low	Close	Volume	OpenInt
0	2010-06-28	17.00	17.00	17.00	17.00	0	0
1	2010-06-29	19.00	25.00	17.54	23.89	18783276	0
2	2010-06-30	25.79	30.42	23.30	23.83	17194394	0
3	2010-07-01	25.00	25.92	20.27	21.96	8229863	0
4	2010-07-02	23.00	23.10	18.71	19.20	5141807	0

查看data有多少筆資料、欄位：

In [26]:

```
print("Data Shape -->", data.shape)
```

```
Data Shape --> (1858, 7)
```

查看data的數學簡介(平均數、標準差、最小值...)

In [27]:

```
data.describe()
```

Out[27]:

	Open	High	Low	Close	Volume	OpenInt
count	1858.000000	1858.000000	1858.000000	1858.000000	1.858000e+03	1858.0
mean	150.389741	152.898737	147.688064	150.355047	4.416508e+06	0.0
std	107.071675	108.490099	105.481665	107.023737	4.244294e+06	0.0
min	16.140000	16.630000	8.030000	15.800000	0.000000e+00	0.0
25%	31.002500	31.732500	30.285000	31.112500	1.283324e+06	0.0
50%	184.440000	188.660000	181.450000	184.850000	3.421026e+06	0.0
75%	231.477500	235.375000	227.772500	230.920000	5.917672e+06	0.0
max	386.690000	389.610000	379.345000	385.000000	3.714989e+07	0.0

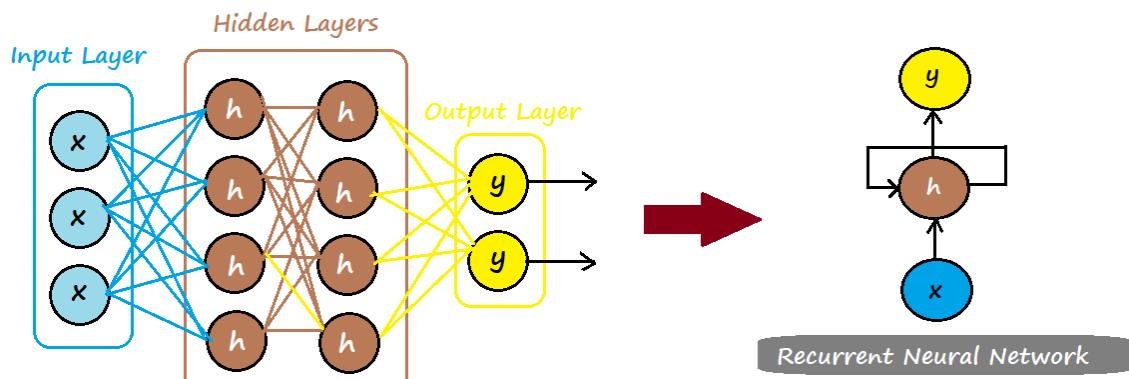
確認data的資料無缺失 → [pandas.DataFrame.isna\(\)](#)

In [28]:

```
print("Do you have a null column? \n", data.isna().sum())
```

```
Do you have a null column?
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
OpenInt   0
dtype: int64
```

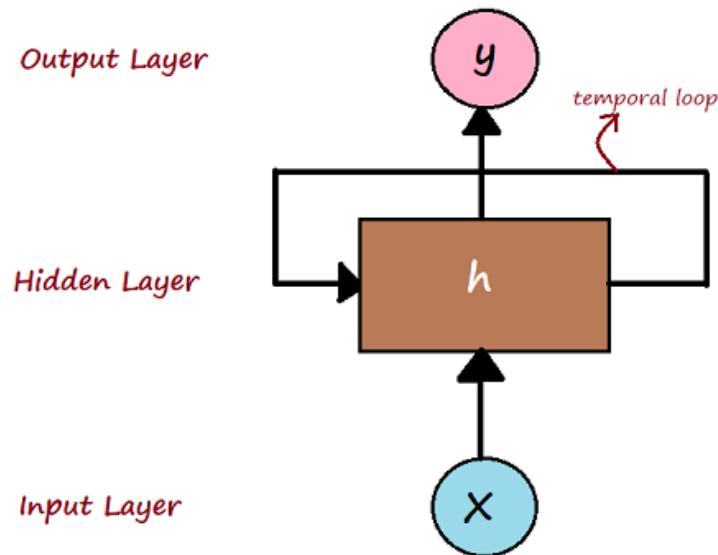
▼ 步驟二：Recurrent Neural Networks (RNN) 介紹



截自該Notebook

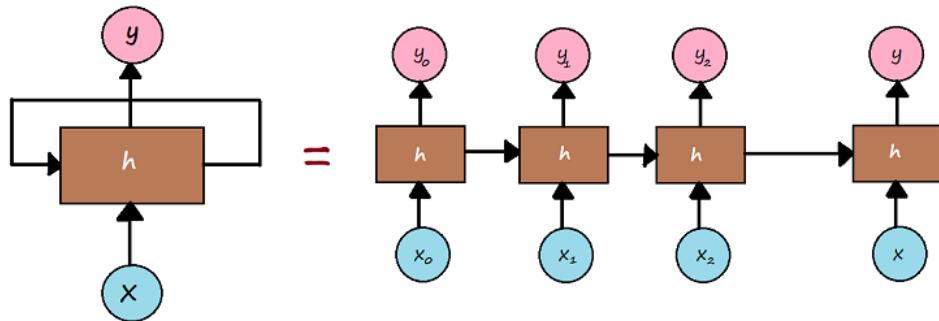
RNN 是一種常用來預測下一步的 Deep Learing 架構，而 RNN和其他 Learng 架構最大的差別在於 RNN具有記憶性，除此之外，其他的 Learng 架構 input 彼此之間是互相獨立的，而 RNN 會使 input 具有相關性，並基於這些 inputs 之間的關係進行學習。

RNN的記憶性來自於其具有類似 loop 的架構：



截自該Notebook

現假設 $t-1$ 時有一個output： y_{t-1} ，有因為有這個 loop 的結構，所以在 t 時刻的 input 會與 $t-1$ 的 output y_{t-1} 一同進入 Hidden Layer 產生 output y_t 。（如下圖所示）



截自該Notebook

因此，RNN的每一次 output 都會與其前一次的 output 有關，或者說，每一個 input 都與前一個 input 有關。

這個特性重要在哪？

我們可以將RNN與FNN(Forward Neural Network)做簡單的比較，在現實世界中，FNN無法處理序列化資料，因為FNN並不會考量前一次的輸入。



序列化資料，Sequential Data，此種資料在時間上具有一定的前後關係，以股價為例，今日的股價肯定與前一天的收盤價有關。

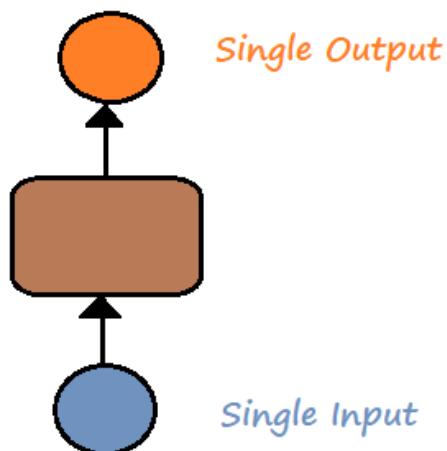
基於上述的特性，RNN可以應用於：

- Image Captioning，圖像描述，透過模型理解圖像結構並利用人類語言描述之。
- Time Series Prediction，時序性資料預測，例如股價預測。
- Natural Language Processing，自然語言解析、語意分析，透過模型理解人類語言的句子意思。
- Machine Translation：從語言A轉換成語言B。

RNN又可以細分成下列幾種：

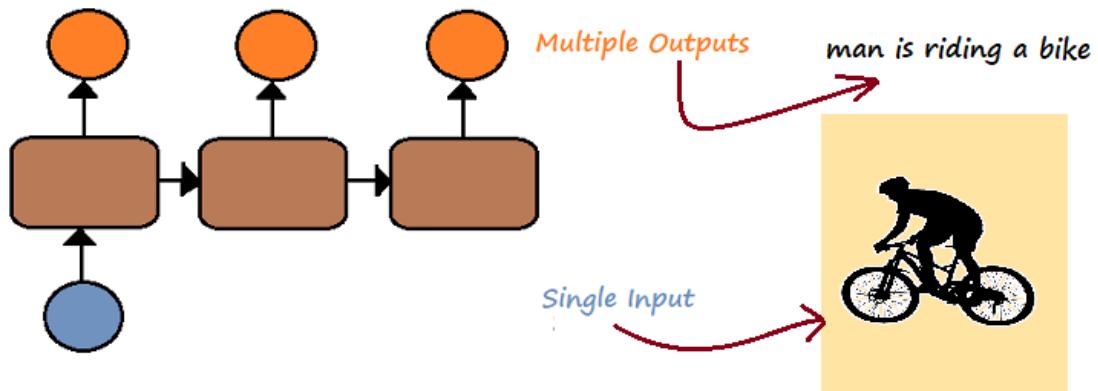
▼ One-to-One

又可以稱做 Vanilla Neural Network，僅具單一 input 和 output，通常用於 ML 問題。



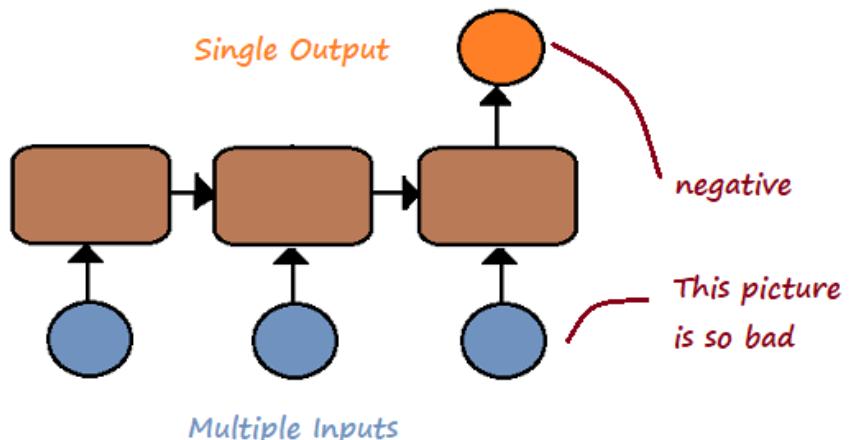
▼ One-to-Many

通常用於圖像描述，以下圖為例，input就是一張圖，output則是一句話描述這張圖。



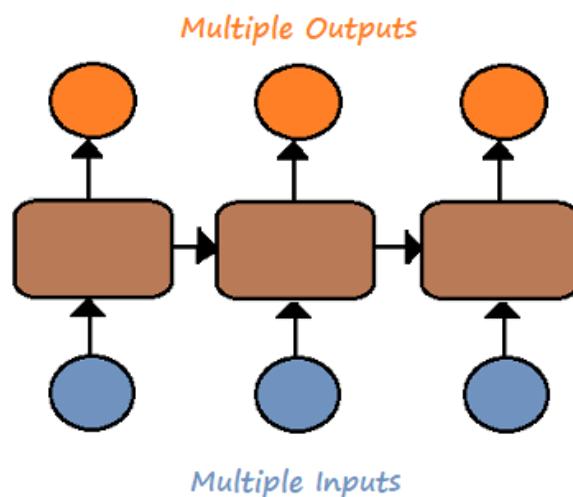
▼ Many-to-One

通常用於語句情緒分析，input是一段話，而output則是這句話的情緒。



▼ Many-to-Many

用於語言轉換，input是A語言，output是B語言。



▼ 步驟三：Data Preprocessing 資料前處理

為了將資料轉換成RNN所接受的格式，因此我們要做資料前處理。

▼ 將資料集切成訓練集和測試集

(通常都是切成 `train:test=8:2`)

```
In [29]:  
training_size = int(len(data)*0.80)  
data_len = len(data)  
  
train, test = data[0:training_size], data[training_size:data_len]
```

```
In [30]:  
print("Training Size --> ", training_size)  
print("total length of data --> ", data_len)  
print("Train length --> ", len(train))  
print("Test length --> ", len(test))
```

```
Training Size --> 1486  
total length of data --> 1858  
Train length --> 1486  
Test length --> 372
```

▼ 正規化資料

為什麼這裡需要做正規化？

- 因為很重要。
- 正規化會使資料的各個特性更兼容，進而提高準確率。
- 在此次模型建構中，由於我們使用股價的開盤價 `open` 作為feature，因此正規化相當適合。

```
In [31]:
# the part of data that we will use as training.
train = train.loc[:, ["Open"]].values

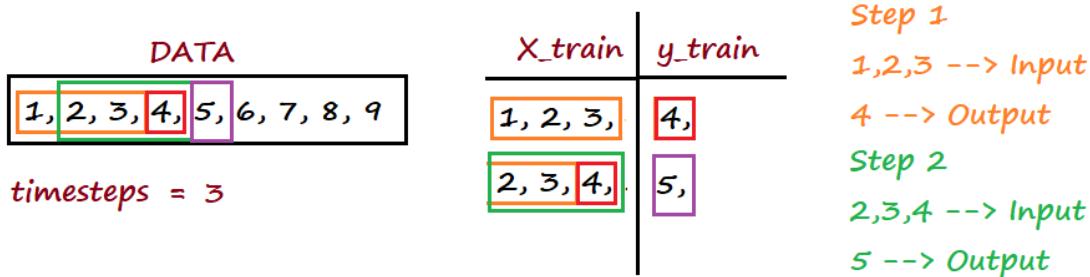
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train)
```

詳細用法參見 [Sklearn.preprocessing.MinMaxScaler](#)

透過 `sklearn` 套件所提供的 `MinMaxScaler` 將整個開盤價的欄位數值縮放至 `[0, 1]`

▼ X_train、Y_train

正規化之後，我們透過下列的方式將訓練集的資料再分成x_train和y_train。



在上圖的例子中，我們一次(一個timestep)抓三筆作為x_train，而實際上我們的timesteps會一次以40為單位來抓，參見下圖。

```
In [32]:
end_len = len(train_scaled)
X_train = []
y_train = []
timesteps = 40

for i in range(timesteps, end_len):
    X_train.append(train_scaled[i - timesteps:i, 0])
    y_train.append(train_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

▼ 重塑資料格式

為什麼需要重塑資料？

- RNN 的建模會希望 inputs 的 Dimensions 是3個維度：
 - Size of Data, 資料的大小。
 - Number of Steps, 一個timestep的大小。
 - Number of Feature, 欲使用的特徵數目 (這裡我們是只抓了 `open` 開盤價作為特徵，所以是1)。

```
In [33]:  
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))  
print("X_train --> ", X_train.shape)  
print("y_train shape --> ", y_train.shape)
```

```
X_train --> (1446, 40, 1)  
y_train shape --> (1446, )
```

▼ 步驟四：利用Keras實作Model

▼ 建立模型

Import keras相關輔助套件

```
In [34]:  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import SimpleRNN  
from keras.layers import Dropout
```

堆疊模型層

```
In [35]:  
regressor = Sequential()  
  
regressor.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True, input_shape = (X_train.shape[1],1)))  
regressor.add(Dropout(0.2))  
  
regressor.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True))  
regressor.add(Dropout(0.2))  
  
regressor.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True))  
regressor.add(Dropout(0.2))  
  
regressor.add(SimpleRNN(units = 50))  
regressor.add(Dropout(0.2))  
  
regressor.add(Dense(units = 1))
```

詳細使用方式可以參見[官方文件](#)。

- units : Dimension of output space, 這裡定義我們所想要的output維度。
- activation : Activation Function, 選擇我們要使用的激活函數, tanh → hyperbolic tangent。

▼ 編譯模型

接著編譯模型

```
In [36]:  
regressor.compile(optimizer= "adam", loss = "mean_squared_error")
```

- optimizer：選擇讓 loss function 逼近最佳解時的方式，`adam` 就是SGD的一種(個人理解)。
- loss：Loss Function，選擇一種 Loss Function 來描述我們的模型在每一次的epoch有多不準(或說有多準)，`mean_squared` 即最小平方法 → 參見官方文件。

▼ Epoch和Batch的大小

- Epoch：1個 Epoch 可以視為在將所有樣本 train 的一次前進(forward pass)或一次後退(backward pass)，也可以說是訓練的週期。
- Batch：Batch Size指的就是在 neural network 中每一層要使用多少個樣本。

```
In [37]:  
epochs = 100  
batch_size = 20
```

▼ Fit模型

將資料餵給建構好、調整完參數的模型

```
In [38]:  
regressor.fit(X_train, y_train, epochs = epochs, batch_size = batch_size)  
  
Epoch 1/100  
73/73 [=====] - 5s 31ms/step - loss: 0.3634  
Epoch 2/100  
73/73 [=====] - 2s 30ms/step - loss: 0.1763  
Epoch 3/100  
73/73 [=====] - 2s 30ms/step - loss: 0.0919  
Epoch 4/100  
73/73 [=====] - 2s 31ms/step - loss: 0.0612  
Epoch 5/100  
73/73 [=====] - 2s 31ms/step - loss: 0.0433  
Epoch 6/100  
73/73 [=====] - 2s 31ms/step - loss: 0.0307  
Epoch 7/100  
73/73 [=====] - 2s 30ms/step - loss: 0.0307  
Epoch 8/100  
  
.  
73/73 [=====] - 2s 31ms/step - loss: 0.0014  
Epoch 99/100  
73/73 [=====] - 2s 31ms/step - loss: 0.0018  
Epoch 100/100  
73/73 [=====] - 2s 32ms/step - loss: 0.0017  
  
Out[38]:  
<tensorflow.python.keras.callbacks.History at 0x7f996cb60e50>
```

▼ 步驟五：Predict

```
In [39]: test.head()
```

	Date	Open	High	Low	Close	Volume	OpenInt
1486	2016-05-24	216.60	218.74	215.18	217.91	2928659	0
1487	2016-05-25	217.91	221.36	216.51	219.58	2514028	0
1488	2016-05-26	220.50	225.26	219.05	225.12	3560997	0
1489	2016-05-27	224.99	225.93	220.75	223.04	3081734	0
1490	2016-05-31	223.04	224.75	221.50	223.23	2046828	0

使用在先前步驟切好的測試用資料集來進行預測

```
In [40]: real_price = test.loc[:, ["Open"]].values
print("Real Price Shape --> ", real_price.shape)
```

Real Price Shape --> (372, 1)

由於我們訓練模型時是使用開盤價 `open` 這個單一特徵，因此預測時也必須是相同dimension。

```
In [41]: dataset_total = pd.concat((data["Open"], test["Open"]), axis = 0)
inputs = dataset_total[len(dataset_total) - len(test) - timesteps:].values.reshape(-1,1)
inputs = scaler.transform(inputs)
```

```
In [42]: X_test = []

for i in range(timesteps, 412):
    X_test.append(inputs[i-timesteps:i, 0])
X_test = np.array(X_test)

print("X_test shape --> ", X_test.shape)
```

X_test shape --> (372, 40)

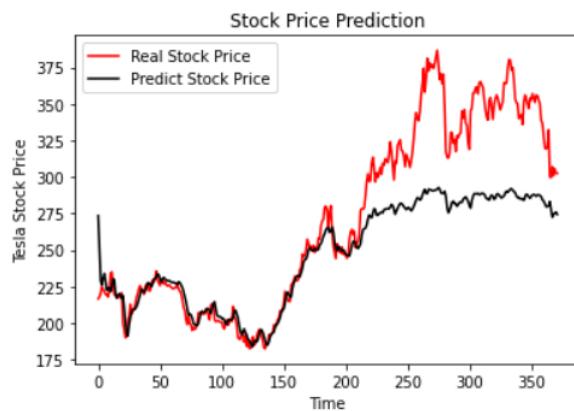
```
In [43]: X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predict = regressor.predict(X_test)
predict = scaler.inverse_transform(predict)
```

最後由於我們在資料前處理時，有將特徵做正規化，因此這裡要反正規化回原本的價格。

▼ 步驟六：Evaluate

最後圖形化預測結果以評估模型的準確率

```
In [44]:  
plt.plot(real_price, color = "red", label = "Real Stock Price")  
plt.plot(predict, color = "black", label = "Predict Stock Price")  
plt.title("Stock Price Prediction")  
plt.xlabel("Time")  
plt.ylabel("Tesla Stock Price")  
plt.legend()  
plt.show()
```



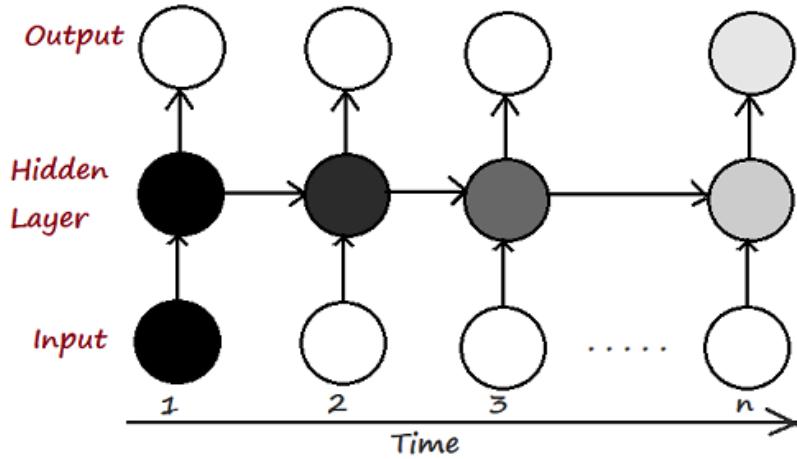
可以清楚看見，使用RNN預測股價時，直到某個時間點都還算準，但至那之後，就幾乎是完全不同的數值了。

Notebook #2 _Long Short-Term Memory N. (LSTM) for Beginners

- Title : Long Short-Term Memory N. (LSTM) for Beginners
- Author : RAFET CAN KANDAR
- Upvoted : 44
- Medal : Silver
- Introduction : 和上一篇是同一個作者，因此作者一樣會先簡短介紹LSTM，應用範圍、為什麼使用LSTM。

▼ Vanishing Gradient Problem, 梯度消失

在上一篇介紹了使用RNN做股價預測，可以從結果得知，在某個時間點之前的預測都還算準確，但至那之後準確率就會大幅下降，而是什麼導致這個情況發生？



Vanishing Gradient Problem 指的就是，在神經網絡反向傳播中，當梯度從後往前傳時，梯度不斷減小，最後趨近0，此時，淺層的神經網絡權重得不到更新，那麼前面隱藏層的學習速率低於後面隱藏層的學習速率，即隨著隱藏層數目的增加，分類準確率反而下降了。→詳細可以參考維基的定義。

以上圖做為範例，將黑色視為 input information，可以看到，在經過一層層的 Hidden Layer 之後，黑色逐漸褪成白色，也就表明最一開始的 input 的資訊逐漸遺失。

▼ Exploding Gradient Problem, 梯度爆炸

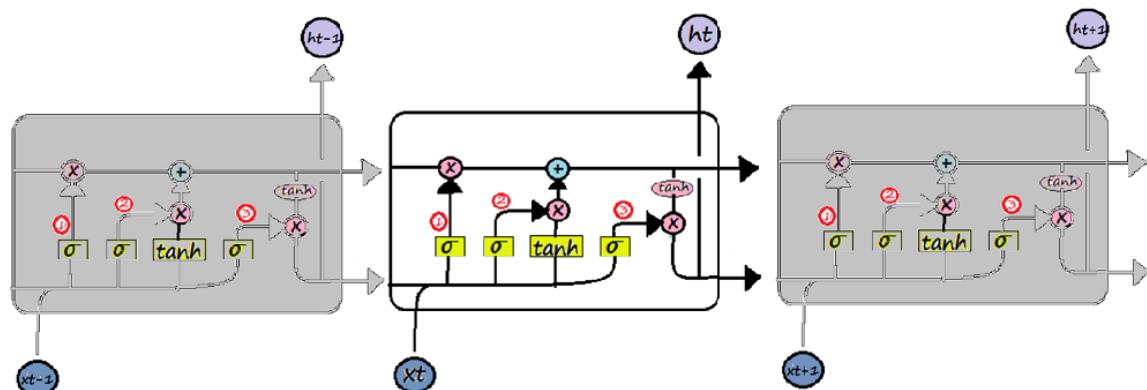
梯度爆炸是一種與梯度消失相反的情況，當進行反向傳播時，梯度從後往前傳時，梯度不斷增大(大於1)，導致權重更新太大，以致於不斷波動，使神經網絡在最優點之間波動。

▼ Long Short-Term Memory Networks, LSTM

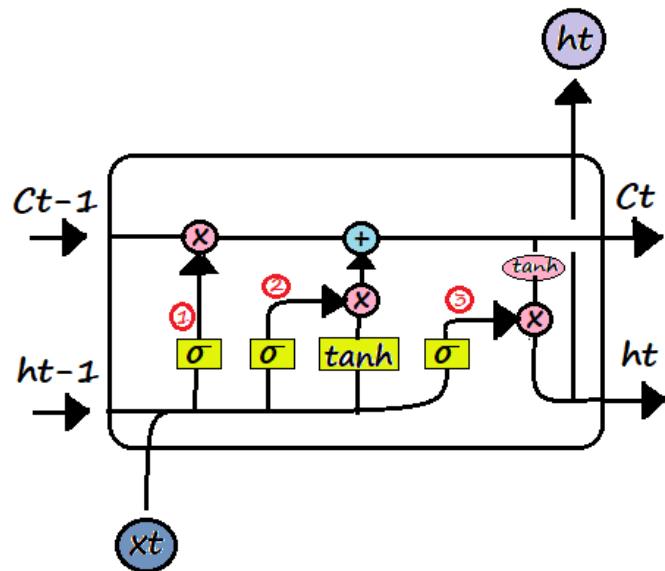
解決梯度消失和梯度爆炸的技巧有下列幾種：

1. 用ReLU、Leaky-ReLU、P-ReLU、R-ReLU、Maxout等替代sigmoid函數。
2. 用Batch Normalization。
3. LSTM的結構設計可以改善RNN中的梯度消失問題。
4. 動態地改變學習率，當梯度過小時，增大學習率，當過大時，減小學習率。
5. 神經網絡的權重標準初始化。

其中最常見且最有效的就屬 LSTM 了。



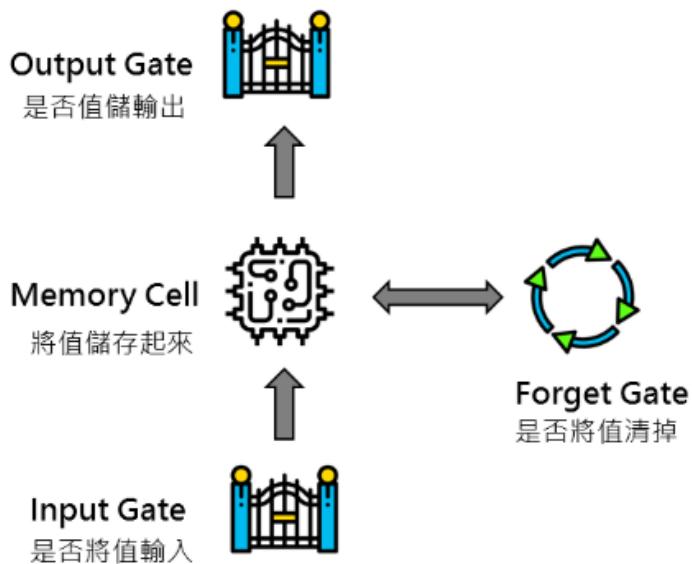
LSTM 可以說是RNN的變化型，在 RNN 中的記憶性通常只到前一層的資訊，而 LSTM 強化了記憶的期限，相較起來可以說 LSTM 具有長期記憶性。



- X 節點代表是否被包括。
- $+$ 節點代表 incoming 的資訊被收集。
- σ 節點可以想成是 sigmoid 層，用來決定記與不記。
- tanh 激活函數，能有效地對付梯度消失。
- h_{t-1} 前一層的 input。
- X_t 該層的 input。
- C_{t-1} 前一層 LSTM unit 中的記憶做為 input 之一。
- C_t 該層需要做記憶以傳遞給之後 Layer 的資訊。
- h_t LSTM 的 output。

LSTM 中3個重要的組件：

LSTM

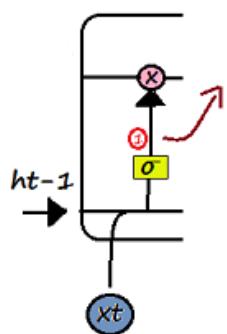


* “是否” 可透過神經網路學習

▼ Forget Gate：是否將值清除

STEP 1 --> Forget Gate

Example:



$ht - 1$ input :

Rafet plays nice ball.

Ilayda is good at swimming.

xt input :

Rafet football plays well. He told me he was the captain because he played well yesterday.

result --> Previous information includes Ilayda and Rafet. The name of Ilayda is not mentioned in the new entry. For this reason, information about Ilayda is deleted and the subject is assigned to Rafet.

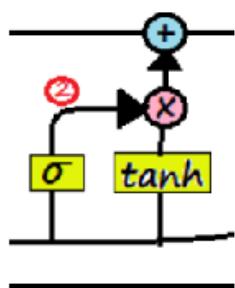
Forget Gate 會決定兩個 input : h_{t-1} 和 x_t 是否要遺忘，以用來只保留有用的資訊。

上圖中舉例， h_{t-1} 有兩個句子，而 Forget Gate 因為此次的 input x_t 只包含其中一個句子的詞語，因此就會判斷將另一個沒有用到的句子遺忘。

▼ Input Gate：是否將值輸入

STEP 2 --> Input Gate

Example:



In the example above. "He told me he was the captain because he played well yesterday". we formed the sentence.

~~He told me~~ --> Less important

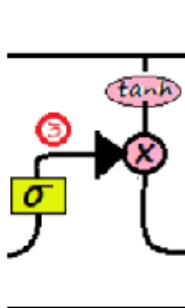
result --> The main thing here is that he plays the ball well and is the captain.

Input Gate 會決定傳進來的 input information 是否要記住，這一個 Layer 由兩個部分組成：sigmoid 以及 tanh，透過這兩個組件可以決定記憶的權重、是否將目前的資訊傳給下一層，如圖中的例子，"He told me"這個部分就被認為是不重要的，因此只將其他的保留。

▼ Output Gate : 是否將值輸出

STEP 3 --> Output Gate

Example:



Let's give this example again starting from above.

Rafet helped his team win because he played so well. Congratulations to who played well.

result --> what would be the best outcome for the space here? The best output for that place will be Rafet.

Output Gate 利用 sigmoid 介於 0~1 的結果來選擇是否要將值輸出。

▼ 步驟一：讀取資料

```
In [41]:  
data = pd.read_csv("/kaggle/input/price-volume-data-for-all-us-stocks-etfs/Stocks/tsla.us.txt")
```

```
In [42]:  
data.head()
```

Out[42]:

	Date	Open	High	Low	Close	Volume	OpenInt
0	2010-06-28	17.00	17.00	17.00	17.00	0	0
1	2010-06-29	19.00	25.00	17.54	23.89	18783276	0
2	2010-06-30	25.79	30.42	23.30	23.83	17194394	0
3	2010-07-01	25.00	25.92	20.27	21.96	8229863	0
4	2010-07-02	23.00	23.10	18.71	19.20	5141807	0

確認資料是否有缺失

```
In [43]:  
print("Do you have a null column? \n", data.isna().sum())
```

```
Do you have a null column?  
Date      0  
Open      0  
High      0  
Low       0  
Close     0  
Volume    0  
OpenInt   0  
dtype: int64
```

▼ 步驟二：資料前處理

為了將資料轉換成LSTM所接受的格式，因此我們要做資料前處理。

由於大多指令都與RNN的情況相同，就不重複一一說明了。

▼ 將資料集切成訓練集和測試集

```
In [44]:  
    training_size = int(len(data)*0.80)  
    data_len = len(data)  
  
    train, test = data[0:training_size], data[training_size:data_len]
```

```
In [45]:  
    print("Training Size --> ", training_size)  
    print("total length of data --> ", data_len)  
    print("Train length --> ", len(train))  
    print("Test length --> ", len(test))
```

```
Training Size --> 1486  
total length of data --> 1858  
Train length --> 1486  
Test length --> 372
```

▼ 正規化資料

```
In [46]:  
    # the part of data that we will use as training.  
    train = train.loc[:, ["Open"]].values  
  
    from sklearn.preprocessing import MinMaxScaler  
    scaler = MinMaxScaler(feature_range=(0, 1))  
    train_scaled = scaler.fit_transform(train)
```

▼ X_train、Y_train

```
In [47]:  
    end_len = len(train_scaled)  
    X_train = []  
    y_train = []  
    timesteps = 40  
  
    for i in range(timesteps, end_len):  
        X_train.append(train_scaled[i - timesteps:i, 0])  
        y_train.append(train_scaled[i, 0])  
    X_train, y_train = np.array(X_train), np.array(y_train)
```

▼ 重塑資料格式

```
In [48]:  
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))  
print("X_train --> ", X_train.shape)  
print("y_train shape --> ", y_train.shape)  
  
X_train --> (1446, 40, 1)  
y_train shape --> (1446, )
```

▼ 步驟三：利用Keras實作Model

▼ 建立模型

LSTM的 model function call 基本上與RNN雷同。

```
In [49]:  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import LSTM  
from keras.layers import Dropout  
  
In [50]:  
regressor = Sequential()  
  
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1],1)))  
regressor.add(Dropout(0.2))  
  
regressor.add(LSTM(units = 50, return_sequences = True))  
regressor.add(Dropout(0.2))  
  
regressor.add(LSTM(units = 50, return_sequences = True))  
regressor.add(Dropout(0.2))  
  
regressor.add(LSTM(units = 50))  
regressor.add(Dropout(0.2))  
  
regressor.add(Dense(units = 1))
```

▼ 編譯模型

```
In [51]:  
regressor.compile(optimizer= "adam", loss = "mean_squared_error")
```

▼ Epoch和Batch的大小

```
In [52]:  
epochs = 100  
batch_size = 20
```

▼ Fit模型

```
In [53]: regressor.fit(X_train, y_train, epochs = epochs, batch_size = batch_size)

Epoch 1/100
73/73 [=====] - 13s 95ms/step - loss: 0.0515
Epoch 2/100
73/73 [=====] - 7s 95ms/step - loss: 0.0067
Epoch 3/100
73/73 [=====] - 7s 93ms/step - loss: 0.0058
Epoch 4/100
73/73 [=====] - 7s 96ms/step - loss: 0.0045
Epoch 5/100
73/73 [=====] - 7s 96ms/step - loss: 0.0050
Epoch 6/100
73/73 [=====] - 7s 94ms/step - loss: 0.0044
Epoch 7/100
73/73 [=====] - 7s 93ms/step - loss: 0.0044
Epoch 8/100

.
.
.

Epoch 99/100
73/73 [=====] - 7s 93ms/step - loss: 0.0014
Epoch 100/100
73/73 [=====] - 7s 92ms/step - loss: 0.0013

Out[53]: <tensorflow.python.keras.callbacks.History at 0x7f7920132710>
```

▼ 步驟四：Predict

準備用切割好的測試集做預測

```
In [55]: real_price = test.loc[:, ["Open"]].values
print("Real Price Shape --> ", real_price.shape)

Real Price Shape --> (372, 1)

In [56]: dataset_total = pd.concat((data["Open"], test["Open"]), axis = 0)
inputs = dataset_total[len(dataset_total) - len(test) - timesteps:].values.reshape(-1,1)
inputs = scaler.transform(inputs)

In [57]: X_test = []

for i in range(timesteps, 412):
    X_test.append(inputs[i-timesteps:i, 0])
X_test = np.array(X_test)

print("X_test shape --> ", X_test.shape)

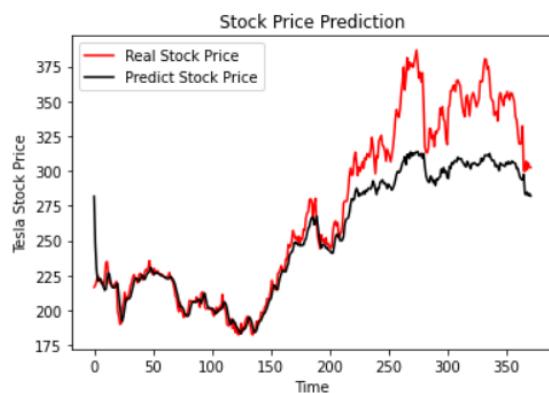
X_test shape --> (372, 40)
```

最後一樣記得反正規化回原本的股價值

```
In [58]:  
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))  
predict = regressor.predict(X_test)  
predict = scaler.inverse_transform(predict)
```

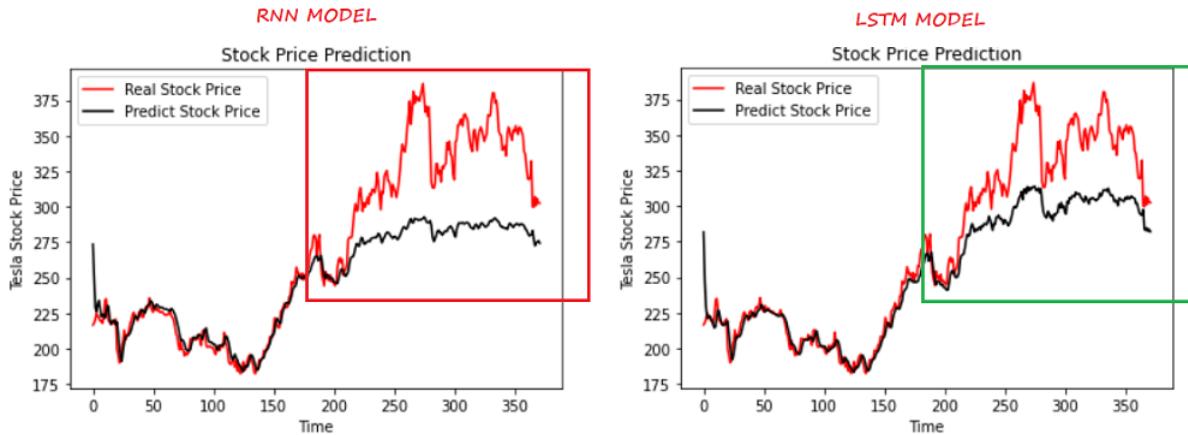
▼ 步驟五：Evaluate

```
In [59]:  
plt.plot(real_price, color = "red", label = "Real Stock Price")  
plt.plot(predict, color = "black", label = "Predict Stock Price")  
plt.title("Stock Price Prediction")  
plt.xlabel("Time")  
plt.ylabel("Tesla Stock Price")  
plt.legend()  
plt.show()
```



可以清楚看見，使用LSTM預測股價時，其漲跌情況大致上已經與實際情況雷同了，只是在價格上仍有不準確的情況。

Conclusion & Insight



我們將 RNN、LSTM 的兩張結果圖放在一起比較，可以清楚看見在漲跌情況上，LSTM 比起 RNN 還要準確許多，而兩者都在價格精準度上有一樣的缺失。

因此在僅考慮漲跌情況來看，對於股價預測，LSTM是比RNN更好的模型。

然而，這兩篇 Notebook 都屬於新手導向，以介紹模型、應用方式及應用情境為主，我們可以在步驟二、三知道這兩次的模型，都只是基於開盤價 open這1個特徵而已。在現實生活中，我們投資股票時，能參考的不是只有技術面的開盤價而已，除了該股的其他特徵之外，或許我們也能透過抓取相關經濟新聞、透過基本面的資本額等等進行分析，在Kaggle上也有另一個股票資料集是與Reddit有關的，所以這兩篇其實在特徵抓取上著墨並不多。

除了特徵抓取之外，資料處理的部分也可以再探討，例如其設置 `timesteps` 一次以 40 為單位，作者也並未說明為什麼是以40為單位進行抓取，另外模型的參數也是一個可能可以增加準確率的議題。

Reference

- Huge Stock Market Dataset

Huge Stock Market Dataset

Historical daily prices and volumes of all U.S. stocks and ETFs

[k https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs](https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs)

Recurrent Neural N. (RNN) Tutorial for Beginners

Explore and run machine learning code with Kaggle Notebooks | Using data from Huge Stock Market Dataset

[k https://www.kaggle.com/rafetcan/recurrent-neural-n-rnn-tutorial-for-beginners](https://www.kaggle.com/rafetcan/recurrent-neural-n-rnn-tutorial-for-beginners)

Long Short-Term Memory N. (LSTM) for Beginners

Explore and run machine learning code with Kaggle Notebooks | Using data from Huge Stock Market Dataset

[k https://www.kaggle.com/rafetcan/long-short-term-memory-n-lstm-for-beginners](https://www.kaggle.com/rafetcan/long-short-term-memory-n-lstm-for-beginners)

- 成交量與持倉量

<https://www.newton.com.tw/wiki/%E6%8C%81%E5%80%89%E9%87%8F>

此種情況在期貨走勢中最常見，多發生在單邊行情開始時期，價位趨勢處於動盪之中。多空雙方對後市的嚴重分歧，形成市場中資金的比拼，但價格此時還未形成統一的整理區間，價格波動幅度快速而頻繁，使短線投資者有足夠的獲利空間。

🔗 <https://www.newton.com.tw/wiki/%E6%8C%81%E5%80%89%E9%87%8F>

35 可取金额	2231.35
00 股票市值	83204.00
35 总资产	85435.35
	修改成
股票余额	可用余额
11000	11000
2200	2200
成本价	保本价
3.265	3.270
市价	盈亏
3.100	-
21.838	21.867
22.320	

持仓量与成交量的关系

从技术分析师的角度来看，这是两个不可或缺的指标。就商品市场的长期因素而言，影响市场价格上涨或是下跌的主要因素是商品的供需关系，然而就短期而言（通常是数小时、数天或数个星期），市场走势往往和基本面完全相反，这是因为在这这么短的时间里，投资决策经常是来自于投资者心理上的研判，时时受希望、恐惧、贪婪和忧虑等因素左右。 ...

知 <https://zhuanlan.zhihu.com/p/39451835>

- 台灣股票資料

歡迎來到 twstock 文件系統! - twstock 1.0.1 說明文件

twstock 是一個簡潔好用的台灣股市程式，透過 twstock，您可以簡單的查詢各類股票之資訊以及即時的股票狀況。twstock 設計僅支援 Python 3 以上之版本，目前最新版 Python 3.6.2 效能已有 Python 2.7 之能力，甚有超過之指標，作者在這邊也鼓勵使用者轉向使用 Python 3。使用上有問題可以透過 issues 或是 twstock gitter.im 請問。如果你正在尋找某個特定指令、class 或 method

🔗 https://twstock.readthedocs.io/zh_TW/latest/

個股日成交資訊TWSE 臺灣證券交易所

臺灣證券交易所全球資訊網介紹公司組織、沿革外，並分為「交易資訊」、「上市公司」、「產品與服務」、「結算服務」、「市場公告」、「法令規章」、「投資人教育」、「統計報表」等，期能提供投資大眾即時完整之訊息，使投資

🔗 https://www.twse.com.tw/zh/page/trading/exchange/STOCK_DAY.html



- Time Series Data

Everything you can do with a time series (stocks)

Explore and run machine learning code with Kaggle Notebooks | Using data from multiple data sources

🔗 <https://www.kaggle.com/neomatrix369/everything-you-can-do-with-a-time-series-stocks/notebook#1.-Introduction-to-date-and-time>



- RNN

循环神经网络RNN介绍1：什么是RNN、为什么需要RNN、前后向传播详解、Keras实现

该文章的主要内容来自：Fundamentals of Deep Learning - Introduction to Recurrent Neural Networks，笔者对该文章进行了翻译、注释、纠错、修改、公式编辑等，使文章更容易理解。让我们从一个问题开始，你能理解下面这句英文的意思吗？"working love learning we on deep"，答案显然是无

知 <https://zhuanlan.zhihu.com/p/32755043>



- 時間序列的深度學習

時間序列的深度學習

這裏提出這樣的問題並不是要從心靈成長雞湯的角度來切入把握時間的重要性。

(當然這也是時間之所以重要的原因之一。) 而是希望從最近流行的深度學習角度來討論"時間"這個特殊維度的重要性。愛因斯坦的狹義相對論認為沒有一個絕對時間。

原文網址：<https://dataholic.wordpress.com/2017/11/18/%E6%99%82%E9%96%93%E5%B8%A8%E5%88%97%E7%9A%84%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92/>



• 梯度消失、梯度爆炸

面试题-梯度消失于梯度爆炸

在反向传播过程中需要对激活函数进行求导，如果导数大于1，那么随着网络层数的增加梯度更新将会朝着指数爆炸的方式增加这就是梯度爆炸。同样如果导数小于1，那么随着网络层数的增加梯度更新信息会朝着指数衰减的方式减少这

原文網址：<https://zhuanlan.zhihu.com/p/51490163>



【QA】什麼是梯度消失(Gradient Vanishing)與梯度爆炸(Gradient Exploded)? - Cupoy

本次想要跟各位探討一下，什麼是梯度消失與梯度爆炸 在深度學習神經網絡中，隨著層數的增加，往往會遇到梯度不穩定的問題，而這個問題便是梯度消失以及梯度爆炸。 ...

原文網址：https://www.cupoy.com/qa/club/ai_tw/0000016D6BA22D97000000016375706F795F72656C65617365515545534



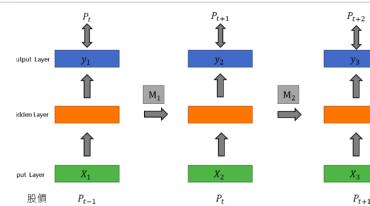
<https://kknews.cc/zh-tw/code/4zagkvx.html>

• LSTM

[Day-16] RNN - LSTM介紹 - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天

今天我們來討論深度學習中，專門在Run時間序列型資料的網路模型 - Recurrent Neural Network (RNN)，在之前所討論到DNN跟CNN模型，兩個均未納入時間序列性質於設計模型。因此，當遇到時間序列問題，大部分都會把時間的性質拿掉，變成特徵獨立的

原文網址：<https://ithelp.ithome.com.tw/articles/10223055>



淺談遞歸神經網路 (RNN) 與長短期記憶模型 (LSTM)

當我們在理解一件事情的時候，通常不會每次都從頭開始學習，而是透過既有的知識與記憶來理解當下遇到的問題；對於語言的理解也是一樣，我們在閱讀一篇文章的時候，通常不是逐字逐句分開了解，而是有辦法從過往知識或是文章的上

原文網址：<https://tengyuanchang.medium.com/%E6%B7%BA%E8%AB%87%E9%81%9E%E6%AD%B8%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-rnn-%E8%88%87%E9%95%B7%E7%9F%AD%E6%9C%9F%E8%A8%98%E6%86%B6%E6%A8%A1%E5%9E%8B-lstm-300cbe5efcc3>

