智慧化企業整合

期末報告

十面霾伏—

以深度學習模型預測新竹市 PM2.5 濃度

第五組

110034544 賴筱庭

目錄

壹、 問題定義

一、 研究動機與目的

空氣中的懸浮微粒會經由鼻及咽喉進入人體，10 微米(μm)以上的微粒可由鼻腔去除，但小於 10 微米(μm)的微粒會經由氣管、支氣管進入人體內部。當懸浮微粒進入肺部，會依不同顆粒大小及化學性質對人體產生不同影響。

許多流行病學研究結果顯示，PM2.5 易附著戴奧辛、重金屬等有害物質，長期吸入可能會引起過敏、氣喘、肺氣腫、肺癌、心血管疾病、肝癌、血液疾病等。且無論長期或短期暴露在高濃度 PM2.5 環境之下，皆會提高呼吸道疾病及死亡的風險，尤其是對於敏感性族群的影響更為顯著。

因此，我希望透過過去空氣品質的資料，來預測未來 PM2.5 的濃度，並提供預警系統。在本次的分析中先將地點鎖定在新竹市，希望能提供民眾以及環保署相關的科研人員在預測時一個依據。

二、 問題分析（5W1H）

表 1 5W1H 問題分析表

Who	民眾、環保署相關科研人員
Where	新竹市
When	欲對 PM2.5 濃度進行預測時
Why	暴露在高濃度的 PM2.5 下，會提高呼吸道疾病以及死亡風險
What	預測 PM2.5 濃度，以提供 PM2.5 預警功能
How	Long Short-Term Memory

貳、 資料預處理

一、 資料集描述

本次分析預測所使用的資料是來自於行政院環境保護署的環境資料開放平台，搜尋關鍵字「空氣品質小時值_XX 縣（市）_XX 站」，即可得到該觀測站的空氣品質資料，包含至 2019 年 3 月至今的數據，觀測站會在每小時觀測並記錄一次各觀測物的濃度。

本次分析以新竹市_新竹站(2021_11)為例，蒐集 2021 年 11 月的資料（共 23516 筆）進行分析。

二、 資料欄位描述

以下為資料欄位的對照表：

表 2 空氣品質資料欄位對照表

SiteId	觀測站代碼
SiteName	觀測站站名
County	觀測站所在縣市
ItemId	觀測物代碼
ItemName	觀測物中文名稱
ItemEngName	觀測物英文名稱
ItemUnit	觀測物單位
MonitorDate	觀測日期、時間
Concentration	濃度（數值）

表 3 空氣品質原始資料

SiteId	SiteName	County	ItemId	ItemName	ItemEngName	ItemUnit	MonitorDate	Concentration
24	新竹	新竹市	1	二氧化硫	SO2	ppb	2021/11/30 23:00	2.1
24	新竹	新竹市	33	細懸浮微粒	PM2.5	μg/m3	2021/11/30 23:00	25
24	新竹	新竹市	7	二氧化氮	NO2	ppb	2021/11/30 23:00	7.1
24	新竹	新竹市	4	懸浮微粒	PM10	μg/m3	2021/11/30 23:00	55
24	新竹	新竹市	3	臭氧	O3	ppb	2021/11/30 23:00	34.7
24	新竹	新竹市	2	一氧化碳	CO	ppm	2021/11/30 23:00	0.33
24	新竹	新竹市	2	一氧化碳	CO	ppm	2021/11/30 22:00	0.37
24	新竹	新竹市	33	細懸浮微粒	PM2.5	μg/m3	2021/11/30 22:00	32
24	新竹	新竹市	7	二氧化氮	NO2	ppb	2021/11/30 22:00	8.2
24	新竹	新竹市	4	懸浮微粒	PM10	μg/m3	2021/11/30 22:00	54
24	新竹	新竹市	3	臭氧	O3	ppb	2021/11/30 22:00	32.6
24	新竹	新竹市	1	二氧化硫	SO2	ppb	2021/11/30 22:00	2.1

三、 資料預處理

在建模前需要先進行資料的預處理，我依序進行了資料整合、資料清理、資料排序、缺失值處理、平穩性檢測、分割資料集、資料正規化以及整理訓練資料。

i. 資料整合、清理以及排序

首先，由於不同月份的資料是分散在不同的檔案，因此要進行整合；接著，將其中「PM2.5」的欄位提取出來；經過上述步驟後，資料的排序會被打亂，且原先檔案的形式是時間靠後的排序會在前，因此要重新排序。

資料預處理

```
[4]  def  data_preprocess():
         from  google.colab  import  files
         uploaded  =  files.upload()
         import  os
         dirPath  =  r"/content/"
         filename  =  os.listdir(dirPath)
         #  將filename按照時間進行排序
         filename.sort(key  =  None,  reverse  =  False)
         #  先將第一筆資料處理完成
         data  =  pd.read_csv(filename[2])
         data  =  data.loc[data['ItemEngName']  ==  'PM2.5']

         for  i  in  range(3,len(filename)):
             temp_df  =  pd.read_csv(filename[i])
             #只選擇pm2.5的raw(資料清理)
             temp_df  =  temp_df.loc[temp_df['ItemEngName']  ==  'PM2.5']
             #將新整理的df與舊的df合併(資料整合)
             data  =  pd.concat([temp_df,data],axis=0)

         #把data倒序，讓時間比較前面的在前面
         data  =  data.iloc[::-1]
         #把dataframe重新排序
         data.index  =  pd.RangeIndex(start=0,  stop=len(data),  step=1)

         data  =  missing_value(data)
         return  data
```

圖 1 資料整合、清理、排序程式碼截圖

ii. 缺失值處理

資料裡有些以 x 表示的缺失值，需對其進行線性補值。

表 4 空氣品質空值欄位

24	新竹	新竹市	33	細懸浮微	PM2.5	μg/m3	2019/3/5 10:00	x
24	新竹	新竹市	7	二氧化氮	NO2	ppb	2019/3/5 10:00	x
24	新竹	新竹市	4	懸浮微粒	PM10	μg/m3	2019/3/5 10:00	x
24	新竹	新竹市	3	臭氧	O3	ppb	2019/3/5 10:00	x
24	新竹	新竹市	2	一氧化碳	CO	ppm	2019/3/5 10:00	x
24	新竹	新竹市	1	二氧化硫	SO2	ppb	2019/3/5 10:00	x

缺失值處理

```
[3]  def missing_value(df):
         #因為缺失值為string(x)的形式，因此先將缺失值轉換為NaN
         for i,item in enumerate(df['Concentration']):
             if item == 'x':
                 df.at[i,'Concentration'] = np.nan
         #將濃度的資料型態轉換為float
         df['Concentration'] = df['Concentration'].astype(float)
         #針對缺失值值進行補值
         df['Concentration'].interpolate(method='linear', inplace=True)
         return df
```

圖 2 缺失值處理程式碼截圖

iii. 平穩性檢測

由於時間序列的數據在處理上需要注意數據是否是平穩的（Stationary），因此，先利用 ADF-test 來檢驗數據的平穩性，若資料為非平穩數據，則會對資料進行差分。

資料檢測

```
def difference(data):
    from statsmodels.tsa.stattools import adfuller
    diff_result = adfuller(data.Concentration)
    if diff_result[1] < 0.05:
        print('The series of concentration is stationary.')
        return data
    else:
        print('The series of concentration is non-stationary.')
        return data.diff()
```

圖 3 平穩性檢測程式碼截圖

iv. 分割資料集

依照比例將資料分割為訓練集以及測試集。

切分資料集

```
[ ] def split_data(data, test_ratio):
        from sklearn.model_selection import train_test_split

        #train : test = 0.7 : 0.3
        #資料不會先進行洗牌
        train, test = train_test_split(data, test_size = test_ratio, shuffle=False)

        #找出其中的Concentration項
        train_set = train['Concentration']
        test_set = test['Concentration']
        return train, test, train_set, test_set
```

圖 4 分割資料集程式碼截圖

v. 資料正規化

資料正規化

```
[ ] def normalize(train_set):
        from sklearn.preprocessing import MinMaxScaler
        #把數據縮放到到[0,1]之間
        sc = MinMaxScaler(feature_range = (0, 1))
        #需將資料做reshape的動作，使其shape為(資料長度,1)
        train_set = train_set.values.reshape(-1,1)
        training_set_scaled = sc.fit_transform(train_set)
        return training_set_scaled, sc
```

vi. 整理訓練資料

將資料整理成時間序列可以用的格式。

整理訓練資料

```
[ ] def training_data_sorting(train_set, training_set_scaled, period_num):
        X_train = []
        y_train = []

        for i in range(period_num,len(train_set)):
            X_train.append(training_set_scaled[i-period_num : i-1, 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))
        return X_train, y_train
```

圖 5 整理訓練資料程式碼截圖

參、 模型介紹與建構

一、 深度學習模型介紹

深度學習（deep learning）是機器學習（machine learning）的其中一個分支，透過多個神經元、隱藏層的組成，可以有效的處理過去無法處理的複雜情況，如：影像辨識、資料預測的領域。

深度學習來自於過去的神經網路（Neural Network），神經網路最早出現在 1943 年，由 McCulloch 和 Walter Pitts 創建，由於過去的電腦處理速度低落，因此該方法並未受到推崇，隨著計算機以及圖形處理速度的發展，以及 Geoffrey Hinton 和 Ruslan Salakhutdinov 發表一篇論文解釋了如何訓練包含多層的神經網絡時，神經網路重新以深度學習面貌出現在人們的視野中。

而基本的神經網路包含三層神經神經元：輸入層（Input layer）、隱藏層（Hidden layer）及輸出層（Output layer）；深度神經網路即為那些擁有多層隱藏層的神經網路。在神經網路的第一層被稱作輸入層，這層會取得外部輸入值，傳入輸入層後就稱為特徵值（feature），如由感測器傳來的圖片，在輸入層的節點不會做任何的運算；神經元數量即為特徵值數量。在輸入層之後的是隱藏層，一個神經網路至少要有一層隱藏層，其作用是學習，隱藏層與學習的好壞不一定成正比，還需依照現實情況進行調整。最後一層為輸出層，位於最後一個隱藏層之後，神經元數量由問題類型來決定。

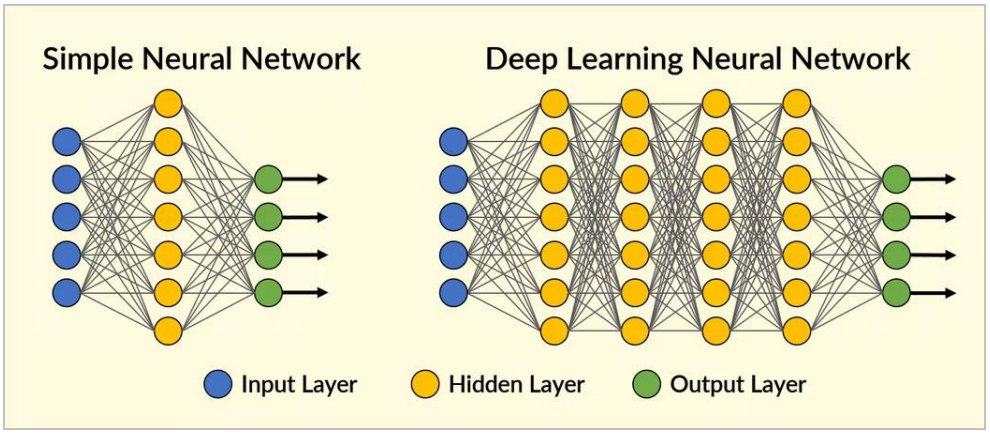

圖 6 神經網路示意圖

i. 循環神經網路（Recurrent Neural Network；RNN）

深度學習中，當資料彼此之間有關聯、順序時，傳統的神經網路由於一次只讀取一個輸入值，造成模型在學習時，無法完整了解資料的樣子。RNN 則是一種具有「記憶力」的模型，其強調資料間前後的關聯性，因此在處理時間序列資料時會有較好的成果。

當隱藏層中的神經元有產生輸出值（在圖中以 y^i 表示）時，會被儲存到 Memory Cell（在圖中以 a^i 表示）裡，當下一個輸入值（在圖中以 x^i 表示）輸入時，模型不只會考慮輸入值，也會考慮 memory 裡的數值，這與時間序列的運作原理極為相似。

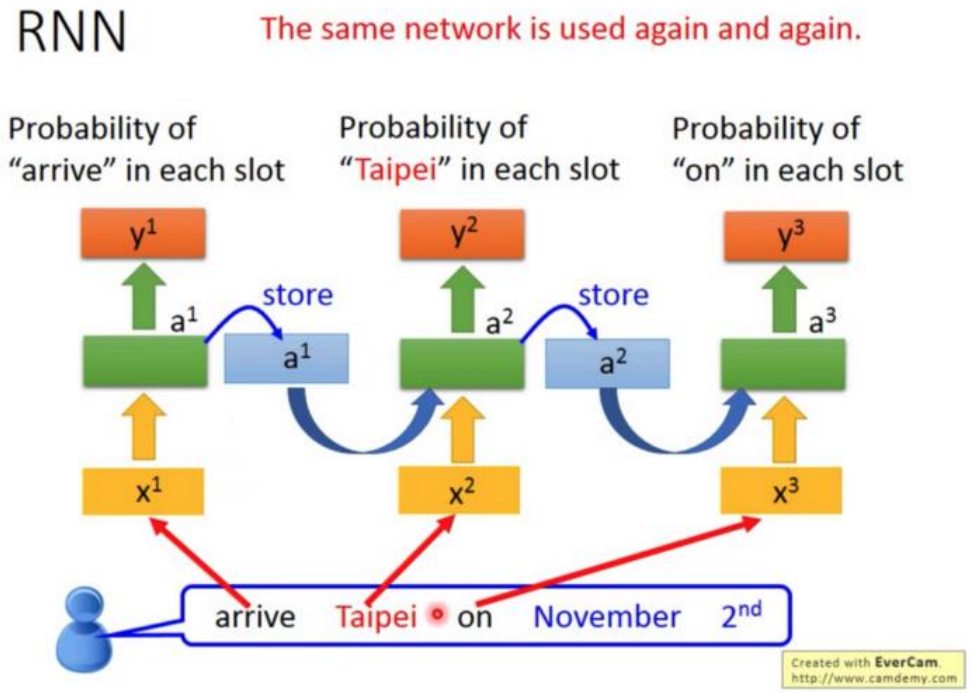

圖 7 遞迴神經網路（RNN）展開圖

然而 RNN 的 Memory Cell 只能儲存一個值，當有下一個輸出值產生時，Memory Cell 就會被覆蓋，因此，我們需要一個可以擁有更長記憶的模型。

ii. 長短期記憶（Long Short-Term Memory；LSTM）

LSTM 是目前 RNN 最常使用的模型，能夠有效解決 RNN 架構梯度消失問題。他在神經單元中加入了輸入閥（Input Gate）、遺忘閥（Forget Gate）、輸出閥（Output Gate），去控制各個值可否被存到 Memory Cell 裡。以下為各閥門以及 Memory Cell 的功能：

表 5 LSTM 之閥門介紹表

名稱	功能	閥門開啟
輸入閥	控制是否將值輸入	輸入值將進入
遺忘閥	是否將 Memory Cell 裡的值清洗	會記得之前保存的值
輸出閥	控制是否將本次的計算結果輸出，若未輸出結果即為 0	可輸出值
Memory Cell	將運算出的數值存起來，提供之後運算使用	X

三個閥門所使用的激活函數通常會使用 Sigmoid Function，這樣做可將值限定在[0，1]之間，當通過激活函數的結果為 0 時，代表閥門關閉，當結果為 1 時，代表閥門開啟。

每個閥門也有其權重（Weight）及偏誤（Bias），可透過機器後續自己去做學習取得。

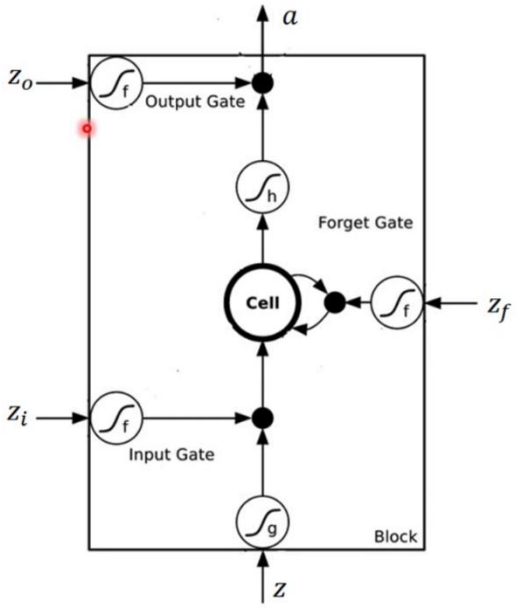

圖 8 長短期記憶(LSTM)之基本架構圖

二、 模型建構

搭建 LSTM 模型。

```
def LSTM_model(train, test, X_train, y_train, sc, period_num, neurons_num, epochs_num, batch_num):
    import keras
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import LSTM
    from keras.layers import Dropout,BatchNormalization
    keras.backend.clear_session()
    regressor = Sequential()
    regressor.add(LSTM(units = neurons_num, activation='relu',input_shape = (X_train.shape[1], 1)))
    regressor.add(Dense(units = 1))
    regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
    history = regressor.fit(X_train, y_train, epochs = epochs_num, batch_size = batch_num)
    plt.title('train_loss')
    plt.ylabel('loss')
    plt.xlabel('Epoch')
    plt.plot(history.history["loss"])
    plt.show()
```

圖 9 LSTM 程式碼截圖

執行整個程式。

```
[13] # run diagnostic experiments
     def run():

         test_ratio = 0.3
         period_num = 24
         repeat = 10

         #LSTM的參數
         neurons_num = 256
         epochs_num = 50
         batch_num = 160

         # 資料預處理
         data = data_preprocess()
         # 檢查平穩、進行差分
         data = difference(data)
         # 切分資料集
         train, test, train_set, test_set = split_data(data, test_ratio)
         # 資料正規化
         training_set_scaled, sc = normalize(train_set)
         # 整理訓練資料
         X_train, y_train = training_data_sorting(train_set, training_set_scaled, period_num)
         # LSTM
         for i in range(repeat):
             test_mse, train_mse = LSTM_model(train, test, X_train, y_train, sc, period_num, neurons_num, epochs_num, batch_num)
             print('%d) TrainRMSE=%f, TestRMSE=%f' % (i, train_mse, test_mse))
```

```
[14] run()
```

圖 10 執行程式碼截圖

我選擇四個可能會模型結果產生影響的參數，進行參數調整，分別是 period_num、neurons_num、epochs_num 以及 batch_num，其分別代表的意義以及欲調整之水準如下：

表 6 LSTM 因子與水準表

變數名稱	代表意義	水準 1	水準 2	水準 3
period_num	將幾個小時的資料視為一個序列，也就是用該序列中第一至倒數第二筆值去預測最後一個值	4	24 (1 天)	48 (2 天)
neurons_num	神經元數量	1	32	128
epochs_num	迭代次數	50	100	X
batch_num	批量大小	40	160	320

接下來將針對這 4 個因子進行全因子實驗，總共需要進行 54 次。

肆、 模型結果

在時間序列預測模型中，由於在意的不是每個預測值是否有完全符合實際值，而是希望找出與實際值差異最小的模型，因此在模型評估上會使用誤差而不是準確率的概念，在這次的分析中我使用「Mean Squared Error」來評估模型的好壞，MSE 越小，代表其誤差越小，模型越好。

一、 參數調整結果

首先，我利用 T 檢定分析 4 個因子的各個水準在信心水準為 0.95 下，是否有顯著差異：

i. Period：3 個水準間皆無顯著差異

```
> #period4 vs period24
> #var.test(group1, group2,conf.level = 0.95)
> t.test(group1, group2, conf.level=0.95,var.eual=T)

        Welch Two Sample t-test

data:  group1 and group2
t = -0.37229, df = 33.544, p-value = 0.712
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.002777742  0.001917964
sample estimates:
  mean of x   mean of y
0.004997222 0.005427111

> #period4 vs period48
> #var.test(group1, group3,conf.level = 0.95)
> t.test(group1, group3, conf.level=0.95,var.eual=T)

        Welch Two Sample t-test

data:  group1 and group3
t = 0.49523, df = 33.098, p-value = 0.6237
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.001547472  0.002543361
sample estimates:
  mean of x   mean of y
0.004997222 0.004499278

> #period24 vs period48
> #var.test(group2, group3,conf.level = 0.95)
> t.test(group2, group3, conf.level=0.95,var.eual=T)

        Welch Two Sample t-test

data:  group2 and group3
t = 0.85907, df = 31.587, p-value = 0.3968
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.001273263  0.003128930
sample estimates:
  mean of x   mean of y
0.005427111 0.004499278
```

圖 11 Period T 檢定結果截圖

ii. Neurons：1 顆與 32、128 顆有顯著差異、32 顆與 128 顆無顯著差異

```
> t.test(group1, group2, conf.level=0.95,var.eual=F)

        Welch Two Sample t-test

data:  group1 and group2
t = 4.1537, df = 17.011, p-value = 0.0006641
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.002104877 0.006449999
sample estimates:
  mean of x   mean of y
0.007831556 0.003554118

> #neurons1 vs neurons128
> #var.test(group1, group3,conf.level = 0.95)
> t.test(group1, group3, conf.level=0.95,var.eual=F)

        Welch Two Sample t-test

data:  group1 and group3
t = 4.1676, df = 17.004, p-value = 0.000645
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.002118864 0.006463692
sample estimates:
  mean of x   mean of y
0.007831556 0.003540278

> #neurons32 vs neurons128
> #var.test(group2, group3,conf.level = 0.95)
> t.test(group2, group3, conf.level=0.95,var.eual=T)

        Welch Two Sample t-test

data:  group2 and group3
t = 0.65208, df = 26.634, p-value = 0.5199
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.973638e-05  5.741612e-05
sample estimates:
  mean of x   mean of y
0.003554118 0.003540278
```

圖 12 Neurons T 檢定結果截圖

iii. Epoch：2 個水準間無顯著差異

```
> #epoch50 vs epoch100
> #var.test(group1, group2,conf.level = 0.95)
> t.test(group1, group2, conf.level=0.95,var.eual=T)

        Welch Two Sample t-test

data:  group1 and group2
t = 0.3454, df = 51.564, p-value = 0.7312
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.001462313  0.002070239
sample estimates:
  mean of x   mean of y
0.005126519 0.004822556
```

圖 13 Epoch T 檢定結果截圖

iv. Batch：3 個水準間皆無顯著差異

```
> #batch40 vs batch160
> #var.test(group1, group2,conf.level = 0.95)
> t.test(group1, group2, conf.level=0.95,var.eual=F)

        Welch Two Sample t-test

data:  group1 and group2
t = -1.7876, df = 25.227, p-value = 0.08587
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.0040044036  0.0002821814
sample estimates:
  mean of x   mean of y
0.004038222 0.005899333

> #batch40 vs batch320
> #var.test(group1, group3,conf.level = 0.95)
> t.test(group1, group3, conf.level=0.95,var.eual=F)

        Welch Two Sample t-test

data:  group1 and group3
t = -1.0484, df = 28.151, p-value = 0.3034
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.0027992388  0.0009035721
sample estimates:
  mean of x   mean of y
0.004038222 0.004986056

> #batch160 vs batch320
> #var.test(group2, group3,conf.level = 0.95)
> t.test(group2, group3, conf.level=0.95,var.eual=T)

        Welch Two Sample t-test

data:  group2 and group3
t = 0.75674, df = 32.897, p-value = 0.4546
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.001542399  0.003368955
sample estimates:
  mean of x   mean of y
0.005899333 0.004986056
```

圖 14 Batch T 檢定結果截圖

接著，排出測試集誤差（Test_MSE）最小的前 10 個參數組合，分別如下：

表 7 誤差最小的前 10 筆

No.	預測期數	**神經元**	Epoch	Batch	Test
1	24	**128**	50	160	0.003466
2	48	**32**	50	320	0.003484
3	48	**32**	100	160	0.003484
4	48	**128**	100	160	0.003487
5	24	**128**	50	320	0.003489
6	24	**32**	50	40	0.00349
7	24	**32**	50	160	0.00349
8	48	**32**	50	160	0.003496
9	48	**128**	100	40	0.003501
10	48	**128**	50	40	0.003507

可以發現，在表現最好的 10 個組合中，神經元確實只有 32 以及 128 兩種水準。

二、 未來濃度預測

在最後，我會利用過去的資料預測未來一個小時的數值，在這份報告中也就是 2021/12/01 00:00 的 PM2.5 濃度，以利使用者可自行進行濃度預測。

```
#預測新的一筆
X_Future = []

j = (len(inputs)+1)
X_Future.append(inputs[j-period_num:j-1, 0])

X_Future = np.array(X_Future)
X_Future = np.reshape(X_Future, (X_Future.shape[0], X_Future.shape[1], 1))
Y_Future_predict = regressor.predict(X_Future)
Y_Future_predict = sc.inverse_transform(Y_Future_predict)
```

圖 15 未來濃度預測程式碼截圖

預測結果如下：

```
The concentration of PM2.5 in Hsinchu will be :  24.132246
```

圖 16 未來濃度預測呈現

伍、 結論

一、目前已完成

i. 可利用模型分析各地的空氣品質資料

ii. 使用者可自行預測未來空氣品質

二、未來展望

i. 建立視覺化平台，方便使用者操作

ii. 自動抓取最新資料，結合平台提供預警功能

iii. 考慮溫度、風向等多變量對空氣品質的影響，建立更完善的模型

陸、 參考資料

屏東縣政府環保局（2021/12/30）. PM2.5 對我們健康有什麼危害？

行政院環境保護署. 空氣品質小時值_新竹市_新竹站

TIBCO. 什麼是深度學習？

iT 邦幫忙（2021/09/18）. Day 12: 人工神經網路初探 深度學習

Ray Bernard（2019/03/26）. SIW. Deep Learning to the Rescue

SarielWang（2020/05/29）.李宏毅教授-RNN 課程影片(Part I)重點整理-Part I https://medium.com/@s716419/李宏毅教授-rnn 課程影片-part-i-重點整理-part-i-3af747baf2db

柒、 附錄

附表為全因子實驗之完整結果，呈現如下：

表 8 全因子實驗結果

No.	預測期數	神經元	Epoch	Batch	Train	Test
1	4	1	50	40	0.014995	0.003573
2	4	1	50	160	0.011499	0.012063
3	4	1	50	320	0.01161	0.012094
4	4	1	100	40	0.014874	0.003589
5	4	1	100	160	0.011381	0.01206
6	4	1	100	320	0.015141	0.003572
7	4	32	50	40	0.014528	0.003588
8	4	32	50	160	0.015085	0.003582
9	4	32	50	320	0.015121	0.00357
10	4	32	100	40	0.015038	0.003573
11	4	32	100	160	0.016036	0.003586
12	4	32	100	320	0.015226	0.003591
13	4	128	50	40	0.014531	0.003573
14	4	128	50	160	0.015442	0.003589
15	4	128	50	320	0.015233	0.003571
16	4	128	100	40	0.014102	0.003657
17	4	128	100	160	0.015414	0.003558
18	4	128	100	320	0.015547	0.003561
19	24	1	50	40	0.011316	0.01205
20	24	1	50	160	0.011792	0.012089
21	24	1	50	320	0.011527	0.012107
22	24	1	100	40	0.015252	0.003556
23	24	1	100	160	0.011683	0.012083
24	24	1	100	320	0.015506	0.003573
25	24	32	50	40	0.014899	0.00349
26	24	32	50	160	0.015005	0.00349
27	24	32	50	320	0.014156	0.003591
28	24	32	100	40	0.016039	0.003555
29	24	32	100	160	0.015851	0.003525
30	24	32	100	320	0.015448	0.003509
31	24	128	50	40	0.016252	0.003509
32	24	128	50	160	0.015112	0.003466

33	24	128	50	320	0.01577	0.003489
34	24	128	100	40	0.0152	0.003537
35	24	128	100	160	0.014902	0.003514
36	24	128	100	320	0.01521	0.003555
37	48	1	50	40	0.015658	0.003582
38	48	1	50	160	0.014925	0.003529
39	48	1	50	320	0.015525	0.003763
40	48	1	100	40	0.015752	0.003542
41	48	1	100	160	0.011409	0.012056
42	48	1	100	320	0.011789	0.012087
43	48	32	50	40	0.015695	0.003513
44	48	32	50	160	0.014686	0.003496
45	48	32	50	320	0.015316	0.003484
46	48	32	100	40	0.01496	0.003793
47	48	32	100	160	0.015586	0.003484
48	48	32	100	320	0.014576	0.003512
49	48	128	50	40	0.014905	0.003507
50	48	128	50	160	0.014503	0.003531
51	48	128	50	320	0.016428	0.003527
52	48	128	100	40	0.015232	0.003501
53	48	128	100	160	0.015742	0.003487
54	48	128	100	320	0.01458	0.003593