資訊領域學習成果

建國中學 林[redacted]毅

目錄

一、 摘要

本篇學習成果內容為我高一、高二在資訊領域所學，以及最終做出較具代表性的作品。在文章的開頭，我會先講述我在資訊領域一路以來的心路歷程。以下的內容則有三個主題：「克拉瑪公式解」、「成績預測」、「大數據應用專題」，除了程式本身以及介紹外，我也會講述我在本主題中學到的技巧、觀念，並闡述當中遇到的困難和解決方式。文章的最後則會附上自身在資訊領域的心得、成長、啟發。

二、程式碼、報告連結

克拉瑪公式解：

報告

解行列式程式碼

克拉瑪公式程式碼

成績預測：

報告（含程式碼）

Tensorflow 操作報告

程式碼

大數據應用專題：

報告

程式碼

三、心路歷程

一開始接觸程式是在國三時，剛好在影音平台上看到了基礎的 javascript 教學，於是就點進去看了幾部，然而時近會考，便沒有往下探究，只學到了環境架設跟輸入、輸出字串，第一次與程式的接觸也就此結束，但這也成為了我日後對程式設計感興趣的契機。

後來高一上的資訊課，老師一開始都在上與密碼學、電腦發展、邏輯有關的內容（在文末的補充資料），即使很有趣，但這卻與我所想的程式設計有點出入，因此我在高一的自主學習時段選擇與同學一起參加一個線上課程，學習 python 的基礎語法(包含資料結構、迴圈、import 功能、list 用法等)，這是我第一次有系統性的學習程式設計。

之後的資訊課，老師開始讓我們用填空的方式，引導我們使用 c 語言來解決問題（如質因數分解、幾A幾B、閏年平年等），一次學習兩種語言雖然在一開始使我有點混亂，但在這些課程中，我也開始學習 c 和 python 語言間的差異和優缺，以及程式設計如何實際應用以解決問題。

到了高二，隨著課程的深入，我開始面對的，不是需要「填空」的題目，而是一個個「開放」的問題，專題課的彭天健老師給了我們許多主題，而我們這組最後選擇了「克拉瑪公式解」作為報告題目（c 語言）。此外，在另一堂科技應用課程內，我則做了需要用到 tensorflow 這個插件的「成績預測」專題（python），在老師提供的架構上，以不斷的實驗去解析大量數據內藏有的規律。最後，則是大數據應用專題，這是我第一個從頭到尾自己寫的專題，雖然過程很艱難，但最後也獲得了不小的成就感。

程式設計，對我而言仍是一塊艱深的領域，這也是我會喜歡程式設計的原因之一，正因為它的廣闊彷彿無窮無盡，所以才有偌大的發展空間，此外，我也很享受這種挑戰所帶來的感覺，每一次解決問題、每一次與同儕合作、每一次學到新的用法和演算法思考方式，都使我豁然開朗。現在回首，在高一、高二的學習路上，我收穫了不少。而往後，我也會承接現在的所學、所思、所想，並繼續向前邁進。

四、 正文

克拉瑪公式解

目標：利用克拉瑪公式解出１～９元１次方程式

我負責的內容：解釋程式、做簡報、口頭報告

環境：Dev c++ , C 語言

報告架構：

1. 克拉瑪公式介紹

2. 解釋行列式程式碼

3. 如何用程式算出９元１次的方程式的解

1.克拉瑪公式解證明：

(1) 二元一次方程式

(1)解二元一次方程組：$\begin{cases} a_1x+b_1y=c_1 \cdots(1) \\ a_2x+b_2y=c_2 \cdots(2) \end{cases}$，其中$x,y$是未知數，

我們使用代入消去法解之

$(1)\times b_2-(2)\times b_1 \Rightarrow (a_1b_2-a_2b_1)x=(c_1b_2-c_2b_1)$

$(1)\times a_2-(2)\times a_1 \Rightarrow (a_2b_1-a_1b_2)y=(c_1a_2-c_2a_1)$

$\Rightarrow$可得$\begin{cases} \Delta\cdot x=\Delta_x \\ \Delta\cdot y=\Delta_y \end{cases}$，其中$\Delta=\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$，$\Delta_x=\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}$，$\Delta_y=\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$。

當$\Delta\neq0$時，方程組恰有一解$(x,y)=\left(\frac{\Delta_x}{\Delta},\frac{\Delta_y}{\Delta}\right)$[兩直線交於一點]

當$\Delta=\Delta_x=\Delta_y=0$，方程組有無限多解。[兩直線重合]

當$\Delta=0$，而Δ_x、Δ_y有一不爲 0 時，方程組無解。[兩直線平行]

(2) 三元一次聯立方程式及延伸

將(3)× $\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$ 得

$$a_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}x+b_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}y+c_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}z=d_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} \quad \cdots\cdots(6)$$

(4)(5)代入(6)，消去y,z

$$a_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}x+b_3\left(-\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}x+\begin{vmatrix} d_1 & c_1 \\ d_2 & c_2 \end{vmatrix}\right)+c_3\left(\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}x-\begin{vmatrix} d_1 & b_1 \\ d_2 & b_2 \end{vmatrix}\right)=d_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$$

整理之後得

$$\left(a_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}-b_3\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}+c_3\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}\right)x=d_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}-b_3\begin{vmatrix} d_1 & c_1 \\ d_2 & c_2 \end{vmatrix}+c_3\begin{vmatrix} d_1 & b_1 \\ d_2 & b_2 \end{vmatrix} \quad \cdots\cdots(7)$$

觀察(7)式，等號左端x的係數中，將a_1,a_2,a_3分別換成d_1,d_2,d_3及成爲右端的式子，

$$a_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}-b_3\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}+c_3\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}=\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

$$d_3\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}-b_3\begin{vmatrix} d_1 & c_1 \\ d_2 & c_2 \end{vmatrix}+c_3\begin{vmatrix} d_1 & b_1 \\ d_2 & b_2 \end{vmatrix}=\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}$$

因此(7)可改寫成 $\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}x=\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}$

同理若令$\Delta=\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$，$\Delta_x=\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}$，$\Delta_y=\begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}$，$\Delta_z=\begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}$

則可得 $\begin{cases} \Delta\cdot x=\Delta_x \\ \Delta\cdot y=\Delta_y \\ \Delta\cdot z=\Delta_z \end{cases}$

(3) 結論：

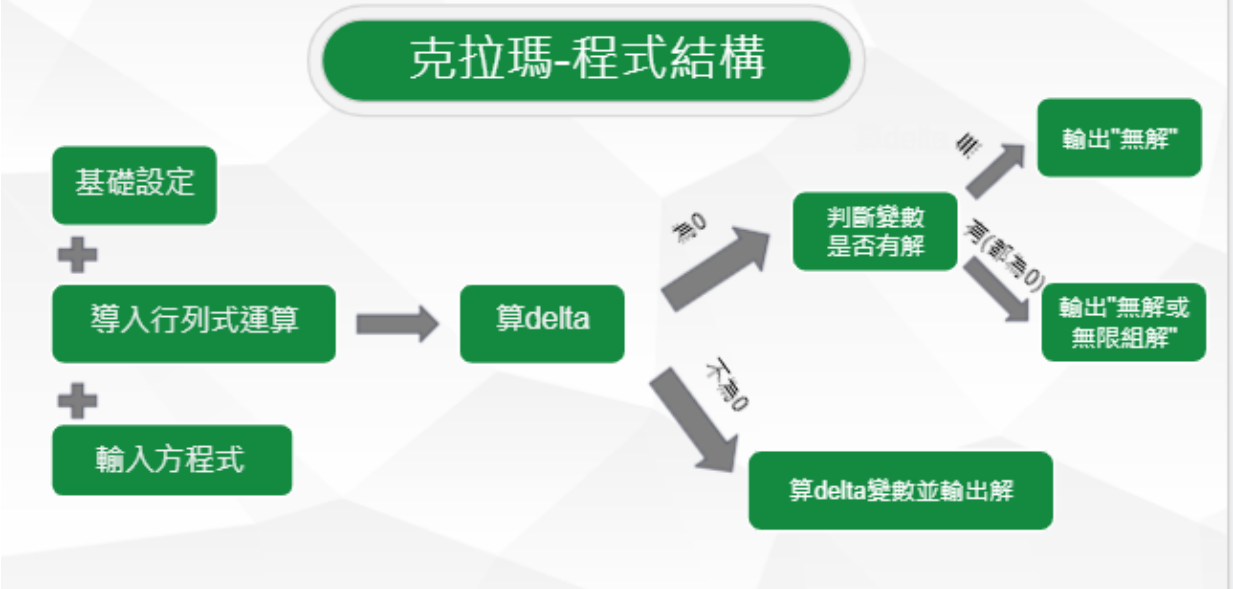
(a)若 $\Delta\neq 0$，則方程組恰有一解：$(\frac{\Delta_x}{\Delta},\frac{\Delta_y}{\Delta},\frac{\Delta_z}{\Delta})$。[克拉瑪公式]

(b)若$\Delta=\Delta_x=\Delta_y=\Delta_z=0$，則方程組無解或無限多解。

(c)若 $\Delta=0$，Δ_x、Δ_y、Δ_z有一不爲 0，則方程組無解。

程式做法

根據上面的結論，我做出了以下這張圖架構圖：



2.行列式運算

根據結論我們要先做出算行列式的程式碼以算 delta 之值，我把程式分為以下四個部分：

基礎設定、參數輸入、輸出結果、主要運算

(1) 基礎設定

```
2   #include <fstream>
3   #include <math.h>
4   #include <stdlib.h>
5
6   using namespace std;
7
8   // 先宣告函式
9   int det(int, int[9][9]);
```

Det(int, int[9][9])就是我們宣告的函式

(2) 參數輸入

```
11  int main() {
12      // n -> 維數; a -> 要算的矩陣
13      int n, a[9][9];
14
15      // input task from file
16      ifstream text("det_test_1.txt");
17
18      text >> n;
19      for(int i = 0; i < n; i++) {
20          for(int j = 0; j < n; j++) text >> a[i][j];
21      }
22      text.close();
```

名詞解釋：

a[9][9]:宣告用以輸入行列式的矩陣
n = 輸入的行列式總共的維數
i, j = 輸入的行列式當前的行、列數

註：我們用 text 文件檔的方式輸入行列式
如：
1 2
4 1 = 1 - (4 * 2) = -7

(3) 輸出結果

```
24      // 輸出答案
25      cout << "The final result = " << det(n, a) << endl;
26
27      //程式暫停
28      system("pause");
29  }
```

解釋：輸出文件檔結果後暫停程式

(4) 主要運算程式

```
30
31  int det(int n, int a[9][9]) {
32      //ans為答案總和  t[9][9]為降階後的行列式
33      int ans = 0, t[9][9];
34
35      // 2階例外處理
36      if(n == 2) return (a[0][0] * a[1][1]) - (a[1][0] * a[0][1]);
37      else {
38          // 沿第一行，從 上列 往 下列 進行降階
39          for(int i = 0; i < n; i++) {
40              // 計算係數以及其正負 (+, -, +, -, +, ...)
41              int coffi = pow(-1, i) * a[i][0];
42
43              // 紀錄 t 矩陣目前待輸入的列數
44              int ty = 0;
45              for(int y = 0; y < n; y++) {
46                  // 如果 ty == 降階所在的列數 -> 跳過這一列 -> ty 不動
47                  if(i == y) continue;
48
49                  // 將 a 矩陣對於第一行第 i 列的係數的 case, 降階後的矩陣
50                  for(int x = 1; x < n; x++) t[ty][x - 1] = a[y][x];
51                  ty++;
52              }
53              // det(n - 1, t)暫時的降階結果, 接下來即可用遞迴進行計算; 圖我另外補
54              ans += coffi * det(n - 1, t);
55          }
56      }
57      return ans;
58  }
```

名詞解釋：

n=階數

det(,)=函式

a[][] = 輸入的行列式

t[][] = 現在降階後的行列式

ans = 行列式的解

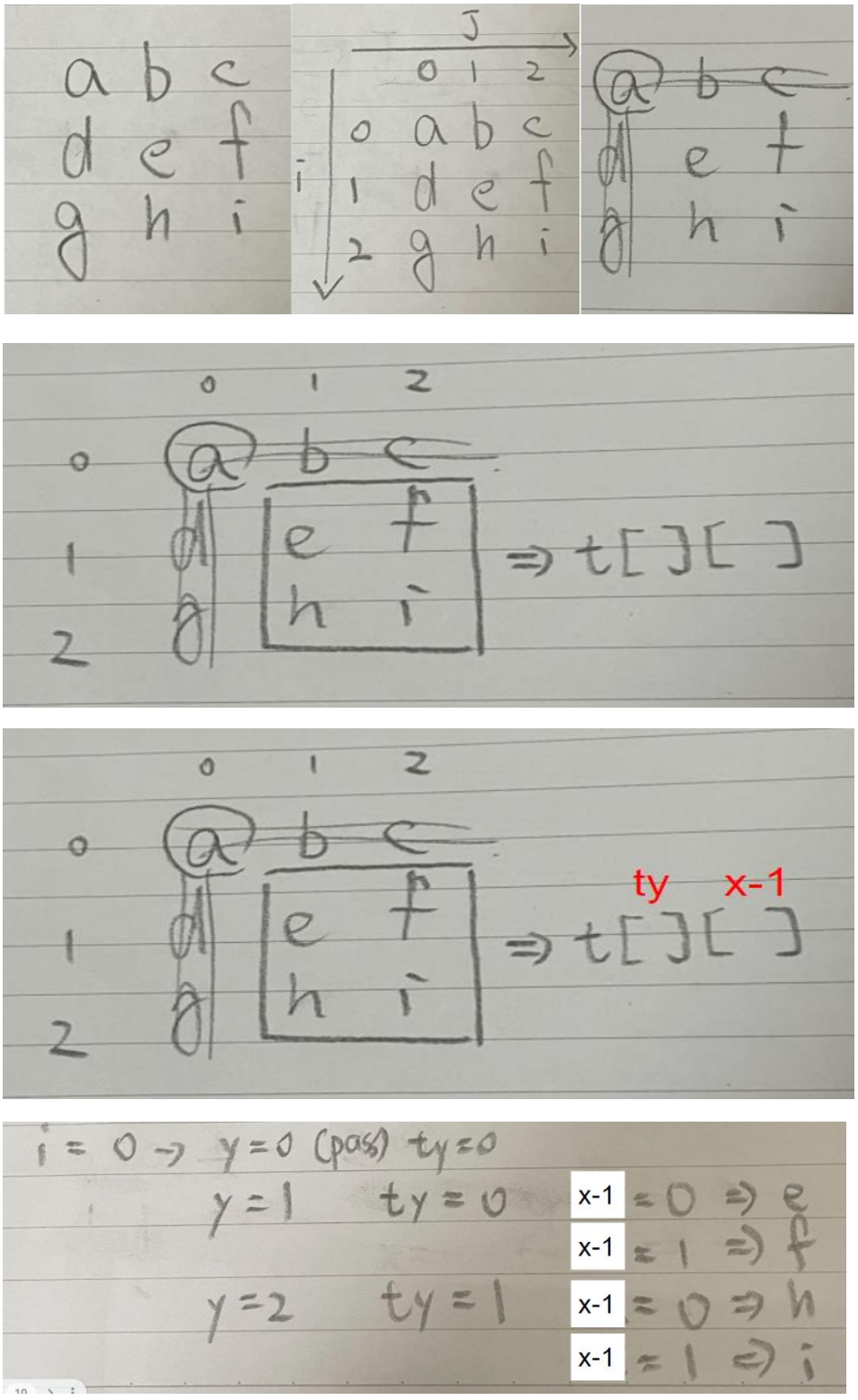
y=用來判斷 t 的行座標是否等於 a 的行座標”i”

ty=t 的行座標

t 的列座標=x-1(往左 1)

說明:我們將所有高於二階的行列式降階，到二階後直接運算（各行解釋在藍色註解）

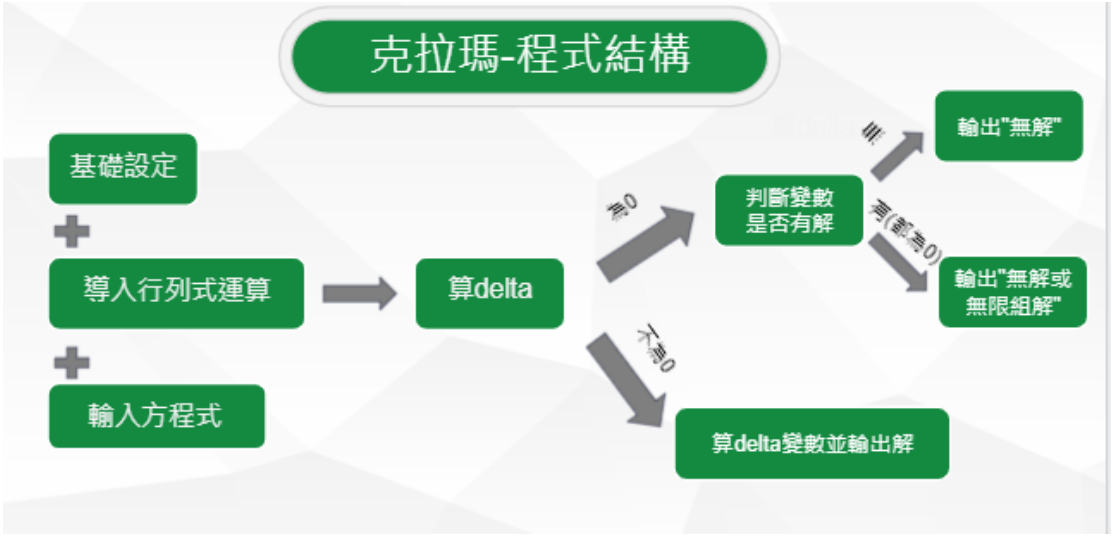
(5)　當時報告時我有畫圖做舉例降階的邏輯（如下）



以三階行列式解釋程式運算邏輯

3.克拉瑪公式解運算

有行列式能算 delta 值後，我們就能使用公式解了



這裡我們會需要這張圖的右半邊，先判斷 delta 是否為 0，若為 0 再判斷 delta x,y,z 是否為 0

程式結構

我把程式結構分為 7 段，分別為：

基礎設定、導入行列式函數、輸入方程式、算 Delta、若 Delta = 0 則判斷有解或無限組解、若 Delta != 0 則算出唯一的解、將答案約分

基礎設定 ➕ 導入行列式運算函式

```cpp
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>

using namespace std;

// 註解詳情請看 - 行列式解說
int det(int n, int a[9][9]) {
    int ans = 0, t[9][9];
    if(n == 2) return (a[0][0] * a[1][1]) - (a[1][0] * a[0][1]);
    else {
        for(int i = 0; i < n; i++) {
            int coffi = pow(-1, i) * a[i][0];
            int ty = 0;
            for(int y = 0; y < n; y++) {
                if(i == y) continue;
                for(int x = 1; x < n; x++) t[ty][x - 1] = a[y][x];
                ty++;
            }
            ans += coffi * det(n - 1, t);
        }
    }
    return ans;
}
```

輸入欲算的方程式

```cpp
int main() {
    // n -> 變數數量; a -> 增廣矩陣
    int n, a[9][10];

    // input task from file
    ifstream text("cramer_test_1.txt");

    text >> n;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n + 1; j++) text >> a[i][j];
    }
    text.close();
```

算D(大delta)


```
48      // try compute D //delta
49      int D[9][9];
50      for(int i = 0; i < n; i++) {
51          for(int j = 0; j < n; j++) D[i][j] = a[i][j];
52      }
53      for(int i = 0; i < n; i++) {
54          for(int j = 0; j < n; j++) cout << D[i][j];
55          cout << endl;
56      }
```

D[][] = 大 Delta 的矩陣（直接複製輸入的 a [][]矩陣）

若D=0之情況(判斷是否有可能有無限組解

```
47      // 存 Di 的行列式值, n = 0, 1, 2, ...
48      int Dn[n];
49
50      if(D_value == 0) {
51          // case 1: D == 0, Dn 存在一個不為 0 -> 方程式無解
52          // case 2: D == 0, Dn 都為 0 -> 無解 || 無限多組解
53          bool exists_not_zero = false;
54
55          // 去算 Di 的行列式值
56          for(int i = 0; i < n; i++) {
57              // Di 的矩陣
58              int Di[9][9];
59              for(int x = 0; x < n; x++) {
60                  // 將第 i 行的元素改為常數的那行元素
61                  if(x == i) {
62                      for(int y = 0; y < n; y++) Di[y][x] = a[y][n];
63                      continue;
64                  }
65                  for(int y = 0; y < n; y++) Di[y][x] = a[y][x];
66              }
67              Dn[i] = det(n, Di);
68              if(Dn[i] != 0) {
69                  exists_not_zero = true;
70                  break;
71              }
72          }
73          if(exists_not_zero) cout << "此方程無解！";
74          else cout << "此方程無解或是無限多組解！";
75      }
```

若D!=0之情況(找出唯一解) ➕ 輸出答案

```
96      else {
97          // 此段皆同上
98          for(int i = 0; i < n; i++) {
99              int Di[9][9];
100             for(int x = 0; x < n; x++) {
101                 if(x == i) {
102                     for(int y = 0; y < n; y++) Di[y][x] = a[y][n];
103                     continue;
104                 }
105                 for(int y = 0; y < n; y++) Di[y][x] = a[y][x];
106             }
107             Dn[i] = det(n, Di);
108         }
```

```
109             cout << "D值為：" << D_value << endl;
110             cout << "此方程各變數解為：" << endl;
111             for(int i = 0; i < n; i++) {
112                 cout << "D" << i + 1 << "值為：" << Dn[i] << endl;
113                 int final_sign = sgn(Dn[i]) * sgn(D_value);
114                 cout << "變數" << i + 1 << "：" << simplify(final_sign, abs(Dn[i]), abs(D_value)) << endl;
115             }
116         }
117         system("pause");
118 }
```

將答案約分

```
29  //將答案約分後輸出函式
30  string simplify(int sign, int a, int b) {
31      if(a == 0) return "0";
32      int Ta = a, Tb = b;
33      while(a != 0 && b != 0) {
34          if(a > b) a %= b;
35          else b %= a;
36      }
37      if(Tb/max(a, b) == 1) return to_string (sign * Ta/max(a, b));
38      return to_string(sign * Ta/max(a, b)) + '/' + to_string(Tb/max(a, b));
39  }
```

4. 結論

在這個專題內，我主要學到了

(1) 以 text 檔方式輸入、輸出結果

(2) 用雙重迴圈將矩陣輸入

(3) 用迴圈原理將大矩陣分解成小矩陣運算

(4) 用畫圖、分解程式碼的方式解說程式

(5) C 語言的語法應用

(6) 小組分工

在這專題中，雖然我負責內容並不是打出程式碼解題，但要將同學的程式碼以報告呈現，勢必要完全了解程式碼中每一步的邏輯和意義。我認為其中最大的困難在於要怎麼將同學打的程式碼，用清晰、淺顯易懂的方式呈現，如其中的程式架構圖、三階行列式一步一步的舉例。我也在同學的程式碼下獲得了許多寶貴的想法和應用實例，同時，我也期望之後持續進步和精進，之後能夠有能力獨立打出這個專題。

5. 資料來源： 彭天健老師、建中學資、同學

Ｔｅｎｓｏｒｆｌｏw應用－成績預測

建國中學　２０８[redacted]

林[redacted]毅

指導老師：黃敦紀老師

一、　基礎介紹

目標:能夠透過老師提供的資料以及 Tensorflow 預測成績

欲預測成績的科目：Tech、Eng　、Math

收入數據：性別等３０餘種

環境：google colab、python

二、　摘要

在這個作業裡面,我們將要利用 Tensorflow 以及一些統計方法預測成績，Tensorflow 是一個大數據學習插件，可以輸入資料給電腦學習，就能預測答案。程式碼是老師提供部分程式碼,要自行找到如何引入資料、刪去不必要的資料並矩陣化、自行印製圖表等。

三、Tensorflow 基礎介紹

Tensorflow 操作報告

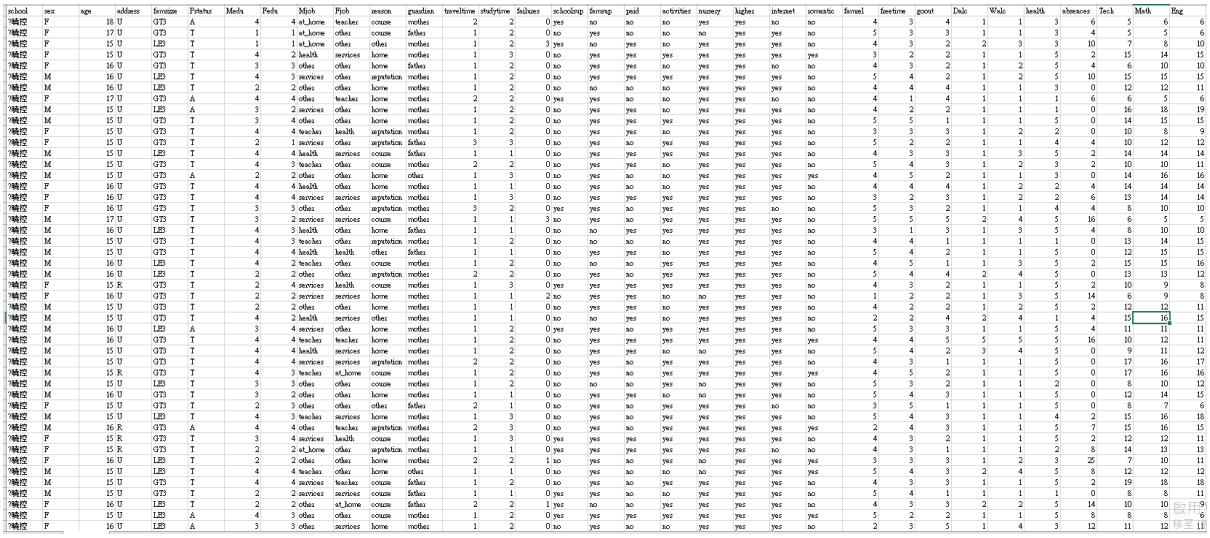
小結：簡單來說，這次會更動三項數值

(1)　Model 數

(2)　迭代值

(3)　訓練資料/測試資料比

四、成績預測程式



school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	Tech	Math	Eng
輔控	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	yes	no	no	no	yes	yes	no	no	4	3	4	1	1	3	6	5	6	6
輔控	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	no	yes	no	no	no	yes	yes	no	5	3	3	1	1	3	4	5	5	6
輔控	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	3	yes	no	yes	no	yes	yes	yes	no	4	3	2	2	3	3	10	7	8	10
輔控	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	no	yes	yes	yes	yes	yes	yes	yes	3	2	2	1	1	5	2	15	14	15
輔控	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0	no	yes	no	yes	yes	yes	no	no	4	3	2	1	2	5	4	6	10	10
輔控	M	16	U	LE3	T	4	3	services	other	reputation	mother	1	2	0	no	yes	yes	yes	yes	yes	yes	no	5	4	2	1	2	5	10	15	15	15
輔控	M	16	U	LE3	T	2	2	other	other	home	mother	1	2	0	no	no	no	no	yes	yes	yes	no	4	4	4	1	1	3	0	12	12	11
輔控	F	17	U	GT3	A	4	4	other	teacher	home	mother	2	2	0	yes	yes	no	no	yes	yes	no	no	4	1	4	1	1	1	6	6	5	6
輔控	M	15	U	LE3	A	3	2	services	other	home	mother	1	2	0	no	yes	yes	no	yes	yes	yes	no	4	2	2	1	1	1	0	16	18	19
輔控	M	15	U	GT3	T	3	4	other	other	home	mother	1	2	0	no	yes	yes	yes	yes	yes	yes	no	5	5	1	1	1	5	0	14	15	15
輔控	F	15	U	GT3	T	4	4	teacher	health	reputation	mother	1	2	0	no	yes	yes	no	yes	yes	yes	no	3	3	3	1	2	2	0	10	8	9
輔控	F	15	U	GT3	T	2	1	services	other	reputation	father	3	3	0	no	yes	no	yes	yes	yes	yes	no	5	2	2	1	1	4	4	10	12	12
輔控	M	15	U	LE3	T	4	4	health	services	course	father	1	1	0	no	yes	yes	yes	yes	yes	yes	no	4	3	3	1	3	5	2	14	14	14
輔控	M	15	U	GT3	T	4	3	teacher	other	course	mother	2	2	0	no	yes	yes	no	yes	yes	yes	no	5	4	3	1	2	3	2	10	10	11
輔控	M	15	U	GT3	A	2	2	other	other	home	other	1	3	0	no	yes	no	no	yes	yes	yes	yes	4	5	2	1	1	3	0	14	16	16
輔控	F	16	U	GT3	T	4	4	health	other	home	mother	1	1	0	no	yes	no	no	yes	yes	yes	no	4	4	4	1	2	2	4	14	14	14
輔控	F	16	U	GT3	T	4	4	services	services	reputation	mother	1	3	0	no	yes	yes	yes	yes	yes	yes	no	3	2	3	1	2	2	6	13	14	14
輔控	F	16	U	GT3	T	3	3	other	other	reputation	mother	3	2	0	yes	yes	no	no	yes	yes	yes	no	5	3	2	1	1	4	4	8	10	10
輔控	M	17	U	GT3	T	3	2	services	services	course	mother	1	1	3	no	yes	no	yes	yes	yes	yes	no	5	5	5	2	4	5	16	6	5	5
輔控	M	16	U	LE3	T	4	3	health	other	home	father	1	1	0	no	no	no	yes	yes	yes	yes	no	3	1	3	1	3	5	4	8	10	10
輔控	M	15	U	GT3	T	4	3	teacher	other	reputation	mother	1	2	0	no	no	no	no	yes	yes	yes	no	4	4	1	1	1	1	0	13	14	15
輔控	M	15	U	GT3	T	4	4	health	health	other	father	1	1	0	no	yes	yes	no	yes	yes	yes	no	5	4	2	1	1	5	0	12	15	15
輔控	M	16	U	LE3	T	4	2	teacher	other	course	mother	1	2	0	no	no	no	no	yes	yes	yes	no	4	5	1	1	3	5	2	15	15	16
輔控	M	16	U	LE3	T	2	2	other	other	reputation	mother	2	2	0	no	yes	no	yes	yes	yes	yes	no	5	4	4	2	4	5	0	13	13	12
輔控	F	15	R	GT3	T	2	4	services	health	course	mother	1	3	0	no	yes	yes	yes	yes	yes	yes	no	4	3	2	1	1	5	2	10	9	8
輔控	F	16	U	GT3	T	2	2	services	services	home	mother	1	1	2	no	yes	yes	no	no	yes	yes	no	1	2	2	1	3	5	14	6	9	8
輔控	M	15	U	GT3	T	2	2	other	other	home	mother	1	1	0	no	yes	yes	no	yes	yes	yes	no	4	2	2	1	2	5	2	12	12	11
輔控	M	15	U	GT3	T	4	2	health	services	other	mother	1	1	0	no	no	yes	no	yes	yes	yes	no	2	2	4	2	4	1	4	15	16	15
輔控	M	16	U	LE3	A	3	4	services	other	home	mother	1	2	0	yes	yes	no	yes	yes	yes	yes	no	5	3	3	1	1	5	4	11	11	11
輔控	M	16	U	GT3	T	4	4	teacher	teacher	home	mother	1	2	0	no	yes	yes	yes	yes	yes	yes	yes	4	4	5	5	5	5	16	10	12	11
輔控	M	15	U	GT3	T	4	4	health	services	home	mother	1	2	0	no	yes	yes	no	no	yes	yes	no	5	4	2	3	4	5	0	9	11	12
輔控	M	15	U	GT3	T	4	4	services	services	reputation	mother	2	2	0	no	yes	no	yes	yes	yes	yes	no	4	3	1	1	1	5	0	17	16	17
輔控	M	15	R	GT3	T	4	3	teacher	at_home	course	mother	1	2	0	no	yes	no	yes	yes	yes	yes	no	4	5	2	1	1	5	0	17	16	16
輔控	M	15	U	LE3	T	3	3	other	other	course	mother	1	2	0	no	no	no	yes	no	yes	yes	no	5	3	2	1	1	2	0	8	10	12
輔控	M	16	U	GT3	T	3	2	other	other	home	mother	1	1	0	no	yes	no	no	yes	yes	yes	no	5	4	3	1	1	5	0	12	14	15
輔控	F	15	U	GT3	T	2	3	other	other	other	father	2	1	0	no	yes	no	yes	yes	yes	no	no	3	5	1	1	1	5	0	8	7	6
輔控	M	15	U	LE3	T	4	3	teacher	services	home	mother	1	3	0	no	yes	no	yes	yes	yes	yes	no	5	4	3	1	1	4	2	15	16	18
輔控	M	16	R	GT3	A	4	4	other	teacher	reputation	mother	2	3	0	no	yes	no	yes	yes	yes	yes	yes	2	4	3	1	1	5	7	15	16	15
輔控	F	15	R	GT3	T	3	4	services	health	course	mother	1	3	0	yes	yes	yes	yes	yes	yes	yes	no	4	3	2	1	1	5	2	12	12	11
輔控	F	15	R	GT3	T	2	2	at_home	other	reputation	mother	1	1	0	yes	yes	yes	yes	yes	yes	no	no	4	3	1	1	1	2	8	14	13	13
輔控	F	16	U	LE3	T	2	2	other	other	home	mother	2	2	1	no	yes	no	no	no	yes	yes	yes	3	3	3	1	2	3	25	7	10	11
輔控	M	15	U	LE3	T	4	4	teacher	other	home	other	1	1	0	no	yes	no	no	no	yes	yes	yes	5	4	3	2	4	5	8	12	12	12
輔控	M	15	U	GT3	T	4	4	services	teacher	course	father	1	2	0	no	yes	no	yes	yes	yes	yes	no	4	3	3	1	1	5	2	19	18	18
輔控	M	15	U	GT3	T	2	2	services	services	course	father	1	1	0	yes	yes	no	no	yes	yes	yes	no	5	4	1	1	1	1	0	8	8	11
輔控	F	16	U	LE3	T	2	2	other	at_home	course	father	2	2	0	yes	no	no	no	yes	yes	yes	no	4	3	3	2	2	5	14	10	10	數用9
輔控	F	15	U	LE3	A	4	3	other	other	course	mother	1	2	0	yes	yes	yes	yes	yes	yes	yes	yes	5	2	2	1	1	5	8	8	8	6
輔控	F	16	U	LE3	A	3	3	other	services	home	mother	1	2	0	no	yes	no	no	yes	yes	yes	no	2	3	5	1	4	3	12	11	12	等至 11

作為數據學習來源與驗證的 csv 檔案

這次作業的個人解讀:

在上次的報告中，我們要輸入的數據有兩種:x 與 y 值，要輸出的則是 c(0 或 1，也就是藍色或紅色)，至於如何判斷顏色的依據，在生成測資的程式中，會生成隨機的點在一個值域中，再用數學不等式去判斷。至於不知道原本程式的 Tensorflow 要怎麼去判斷呢?就我的理解，它會做出一個方程式符合要學習的測資，並在這過程中不斷地修正到盡量貼合原本的條件，而這個方程式，會把每個點作出一個類似加權平均的結果，再用不等式判斷。

例如: x*0.1 + y*0.9 如果大於 0.5 則輸出紅色，反之不大於 0.5 則輸出藍色。

因此這次的結果就是把 30 餘種數據帶入，做類似的數學算式，並輸出結果的分數我們先直接跑一次程式看看，當作對照組 (迭代數 100 / model 100*5 / 分割 25%資料做為測試)

(也就是這些地方會改，第一次展示出來之後不會)

```
#  分割測試、訓練資料
#  分割測試、訓練資料：3/4  拿來訓練、1/4  拿來測試訓練結果
train_i,  test_i,  train_o,  test_o  =  train_test_split  (
        data.drop  (['Tech','Math','Eng'],  axis=1),
        data[['Tech','Math','Eng']],        #  結果0
        test_size  =  0.75  #測試資料占  1/4
)
```

```
#模型初始化
model  =  Sequential()
#類神經網路建構
model.add(Dense(10,  input_dim=56))  #imput  dim
model.add(Dense(100,activation="relu"))
model.add(Dense(100,activation="relu"))
model.add(Dense(100,activation="relu"))
model.add(Dense(100,activation="relu"))
model.add(Dense(100,activation="relu"))
model.add(Dense(3))       #神經元數  #important
```

```
#模型訓練
history  =  model.fit(train_i,train_o,          #輸入與輸出
                      epochs  =100  ,            #迭代數
                      verbose  =  1,             #顯示訓練過程
                      validation_split  =  0.25,  #驗證資料分割比例
)
```

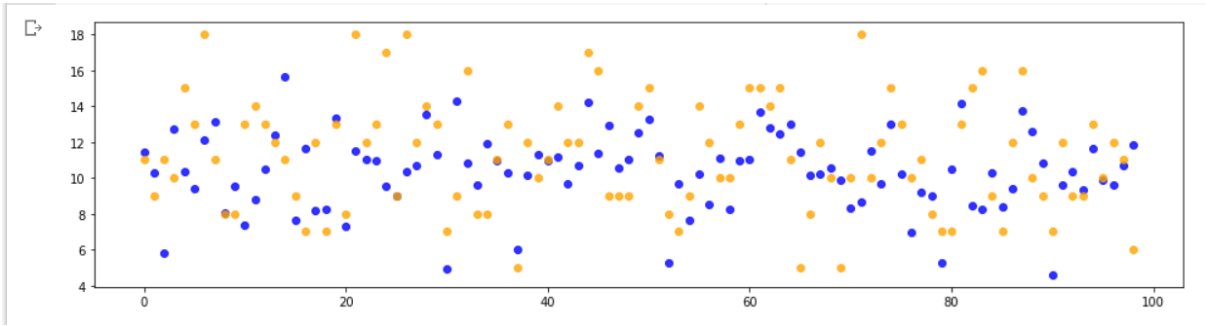
圖的意義:

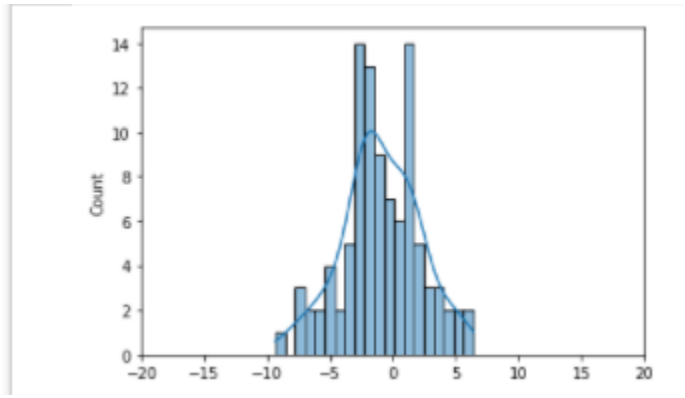
黃色點與藍色點分別為正確的結果和預測的結果，下方則是預測結果與正確結果的差距分布圖。

輸出結果:

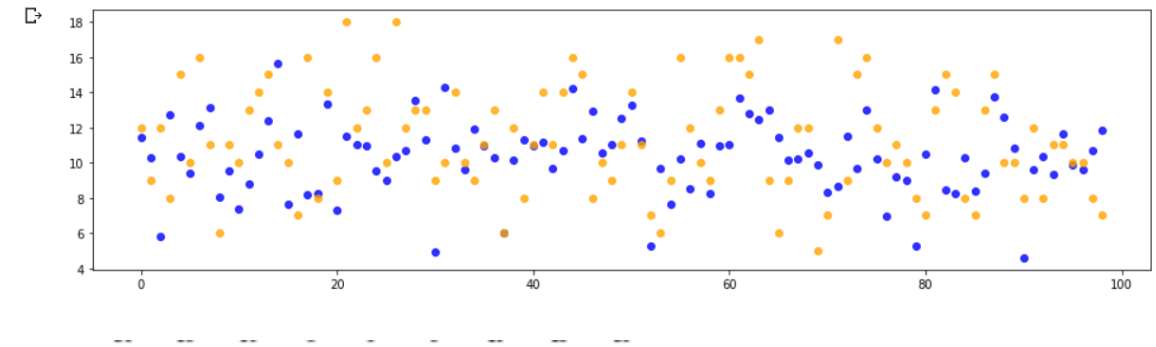
```
train loss: 9.471969604492188
test loss: 22.075729370117188
```

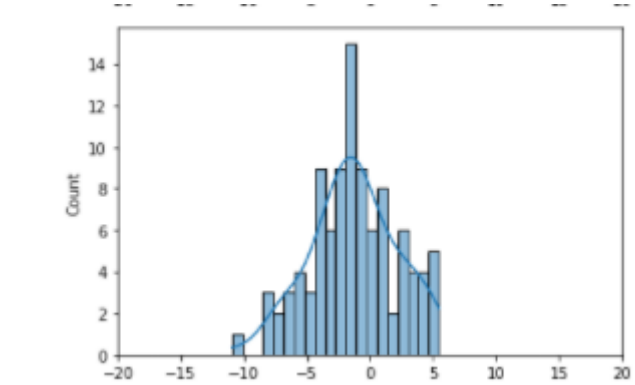
數學 math:



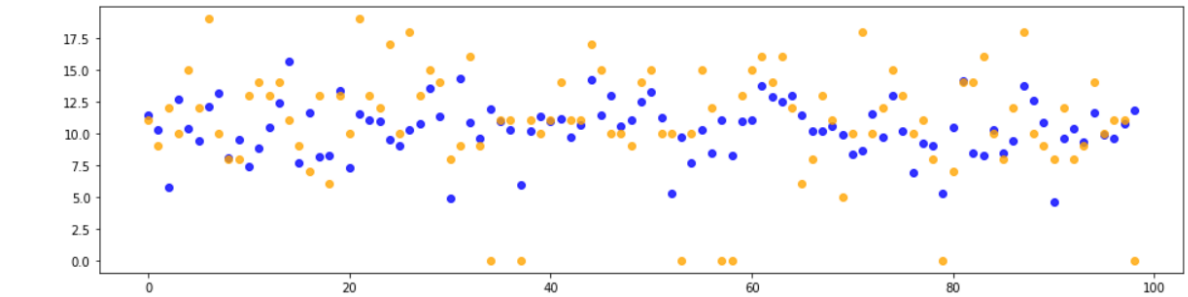


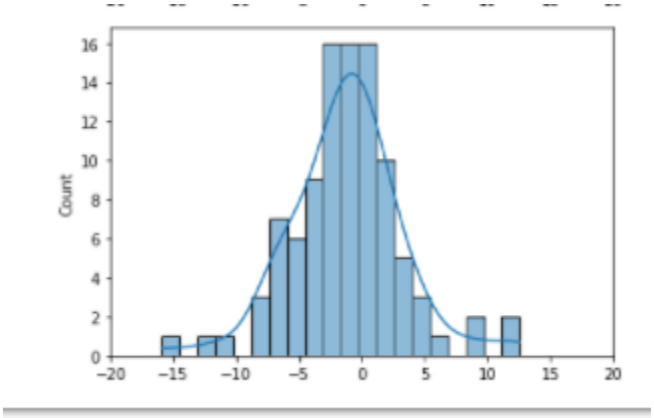
工程 tech:





英文 eng:





五、 測試內容

為甚麼這麼不準?

在圖中我們可以看出很多點遠遠偏離了數據，loss 值更是高達了 9 與 22，依據上次作業 loss 都在 0.1 以內的經驗，可以看出來這次的判斷非常不準，除了類神經網路建構、迭代數、測資數遠遠低於上次外(這次總數據只有 400 筆左右，上次是至少 1000)，還有可能的原因就是這次的資料輸入種類太過於複雜，以我們從小到大的實驗經驗中，都可以看出操縱變因太多會導致結果會難以判斷以致結果混亂，這次輸入的資料種類就像是這種情形。我想，這也是這次會很難測的、結果不準的主因。

那要怎麼降低呢?

我們根據上次的經驗法則推測出可能的解決方案

操縱變因依據:

在上次 Tensorflow 的作業中，我得出來三個方式可以把 loss 值降低的方式

(1)將 model(類神經網路建構)數量改變(增加)

(2)將 epoches(迭代數)增加

(3)增加測資數量

這次由於數據有限，不能無限生測資，因此這次會將增加測資數量改成"用來訓練以及用來測試的數據數量"，以求在其中找出差別。

由於從直方圖看，”Tech”的 loss 值較小(判斷依據:0 的佔最大多數、極端值數字較小也較少)因此接下來會都用 Tech 作為實驗對象

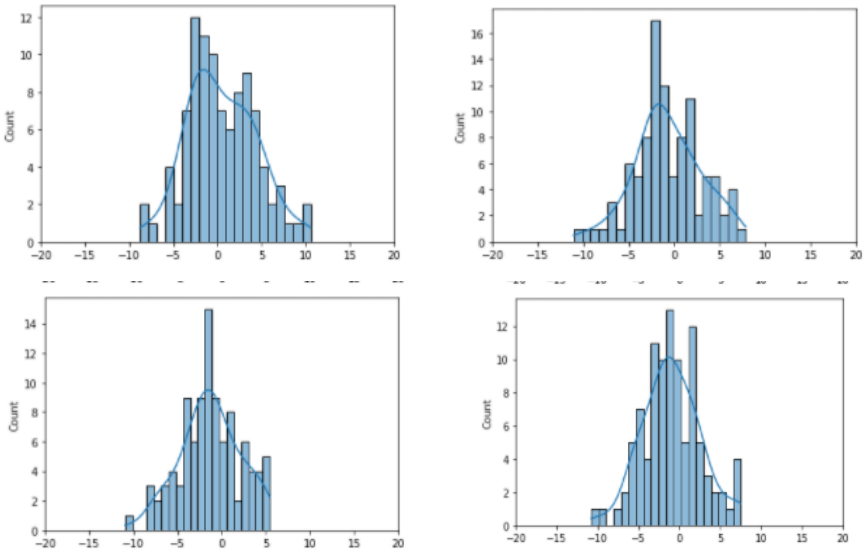
(1)改變 model

```
[59] #模型初始化
     model = Sequential()
     #類神經網路建構
     model.add(Dense(10,  input_dim=56))  #imput  dim
     model.add(Dense(100,activation="relu"))
     model.add(Dense(100,activation="relu"))
     model.add(Dense(100,activation="relu"))
     model.add(Dense(3))       #神經元數  #important
```

示意圖(model300):

*若沒有特別標記則是左到右:300/500/700/1000

300/500/700/1000

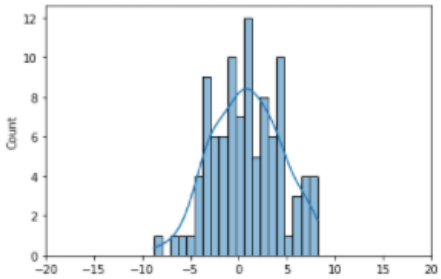


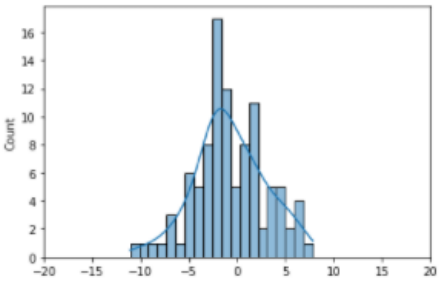
可以看到效果不顯著，而且有趣的是，loss 值竟然會因為類神經網路變少，為了確認是否真的是這樣，我將 300 與 1000 再做了一次:

300 / 1000

```
train loss: 9.439475059509277
test loss: 13.192375183105469
```

```
train loss: 6.963295459747314
test loss: 15.231745719909668
```





結果還是一樣(以 test_loss 來說，由於結果是用測試，所以我會採信這個)，後來我又多做了幾次，皆是同樣的結果，目前我還不知道原因為何，不過我會先採用 model=300 的這個設定。

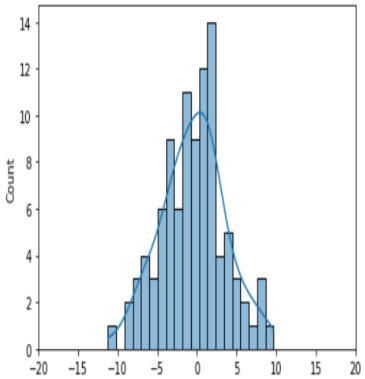
(2)更多 epoches(迭代數)

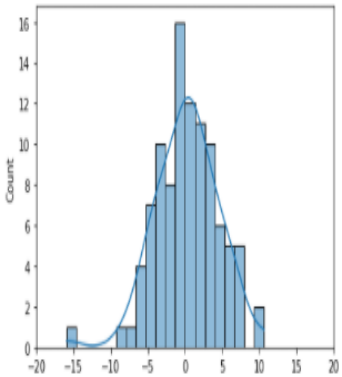
500 / 1000 / 5000

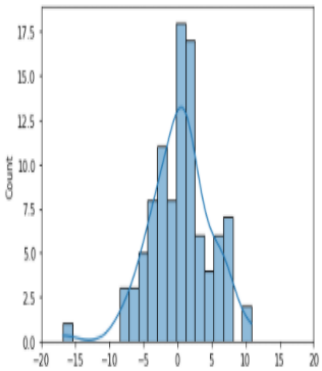
```
train loss: 5.578423500061035
test loss: 22.79547691345215
```

```
train loss: 4.9388556480407715
test loss: 21.310699462890625
```

```
train loss: 4.285490989685059
test loss: 19.229061126708984
```

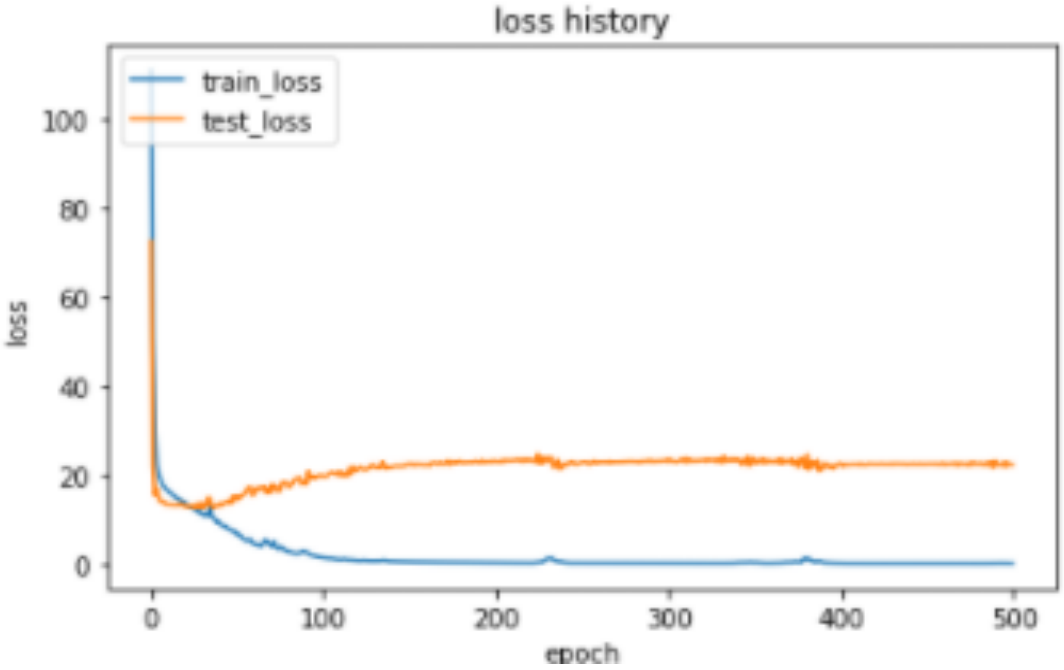






以 loss 來說，很明顯結果是迭代數越高會越準，但 5000 筆迭代的確需要不少時間成本巨大，這一次大概要跑 10 分鐘，可以看出當現實應用中迭代很耗資源，下手前要評估。

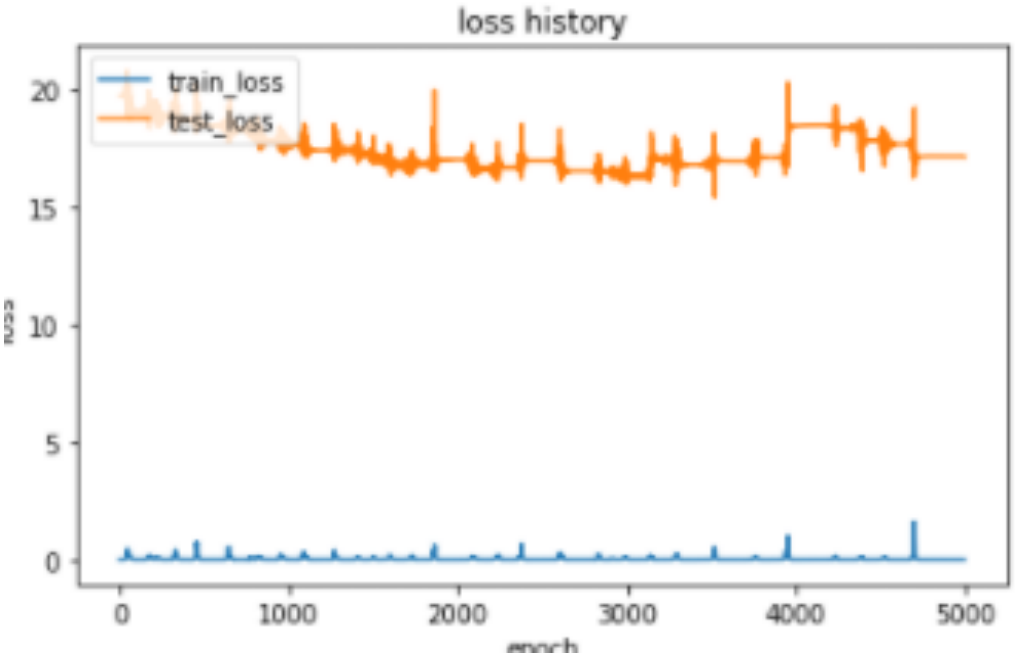
在這次的迭代的過程中，loss 值變化表有幾個特別有趣的事情，我有兩個特別的發現:



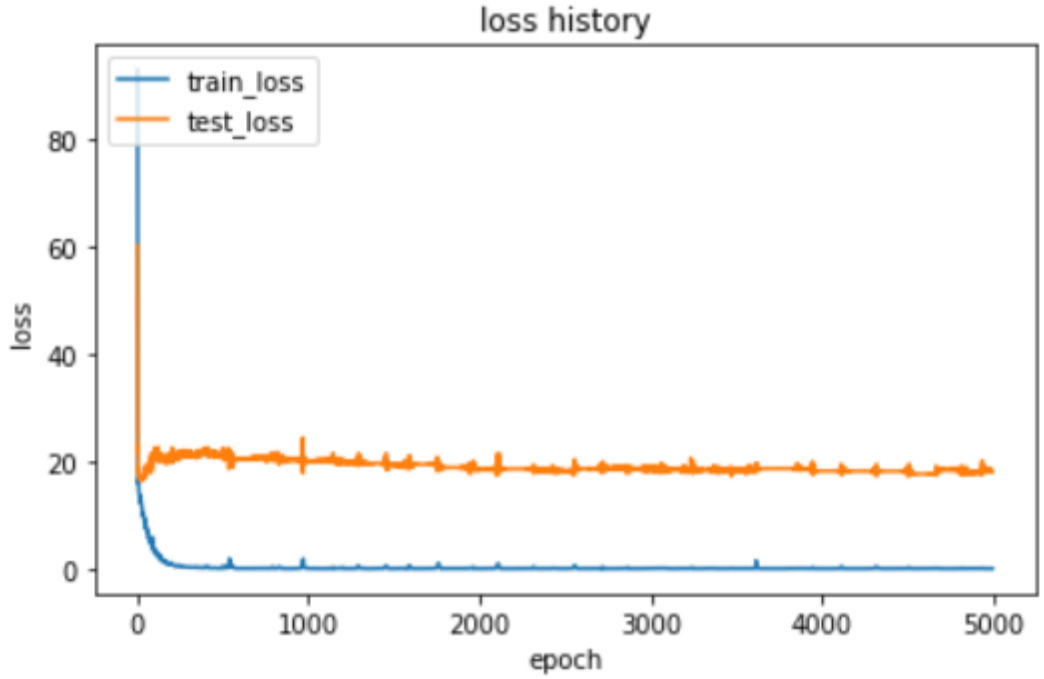
第一、test_loss 會先低再高，以 epoches(300)為例:(上圖)

可以看到 train_loss 會越來越低，但 test_loss 卻掉到最低的地方後又往回跑，接著趨於一個穩定值，這可能代表這個穩定值是在這個情況下，test_loss 的最低底線值，在這之後，loss 值可能就不能再有多大的改變了。

第二、loss 值會有週期性的波動，以 epoches(5000)為例:



在圖中可以看到，在 700、1900、2800、4300 等迭代數中出現了大幅度的降低-升高情況，目前還不知道原因為何，不過可以確定我們必須避開 loss 值的頂峰、並抓到它快速升高之前最低的 loss 值處。為了確認這個大幅度波動會不會每次都不一樣，我再次做了 5000 的迭代:



可以看到每次大幅度擺動的震幅與波動位置都不一樣，因此要如何避免還是個未知數。

(3)改變訓練/測試資料量比

用來測試的資料比:25% 50% 75%
(由於迭代 100 過少，可能會造成結果有大偏差，因此改成 300)

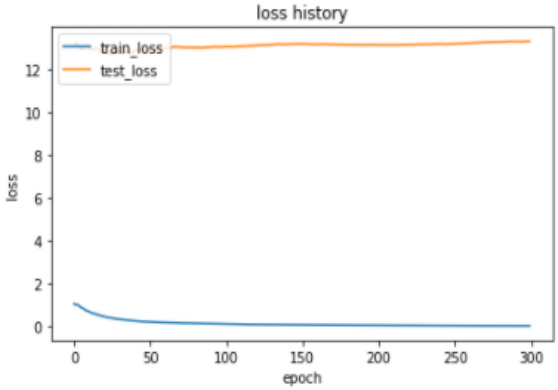
25%	50%	75%
train loss: 25.333852767944336 test loss: 31.237272262573242	train loss: 6.177903652191162 test loss: 21.65949249267578	train loss: 9.64320182800293 test loss: 22.47547721862793

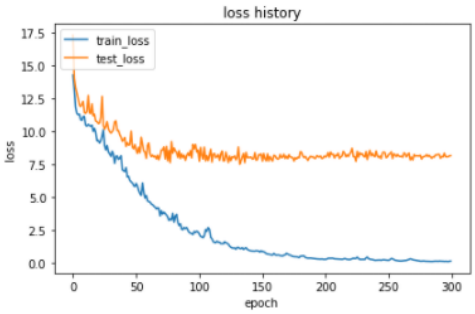
這個結果很合理，就我的判斷，在資料只有 400 筆的情況下，不太能做出可靠的數據推測，要是測試的資料，更何況用來訓練的資料變成 200 筆、100 筆甚至更少，因此做出這結果是在我的預期範圍之內。但是反之，若用來測試的資料量太少(如 10 筆 20 筆)，則無法做出可信任的預測結果。因此，我會把用來測試的資料調成 25%。

六、發現

(1)、

test 資料佔 75%時，這次的 loss 值變動非常平緩，幾乎沒有如同之前的巨量波動，但 25%時卻會劇烈擺動，這是之前的測試中都沒有見過的。(如下圖)





如果每次都能像左邊一樣平滑、沒有劇烈擺動，則可以讓每次的測試結果更加穩定。這可能是之後的目標。

(2)、

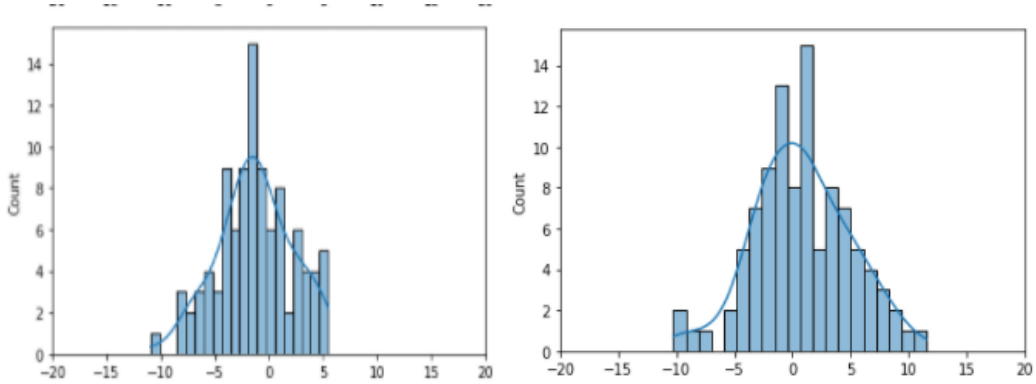
綜合以上的結果，我們將各組實驗組具有最低 loss 值的結果合在一起，並與最開始的對照組比較

```
train loss: 9.471969604492188
test loss: 22.075729370117188
```

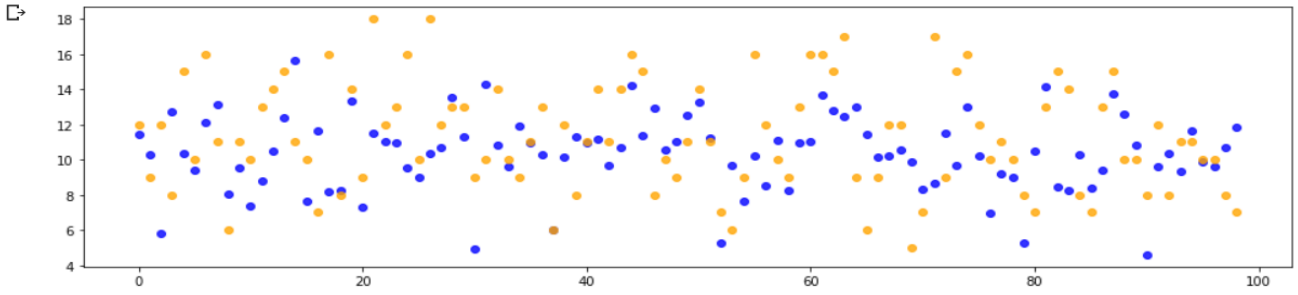
:

最終版

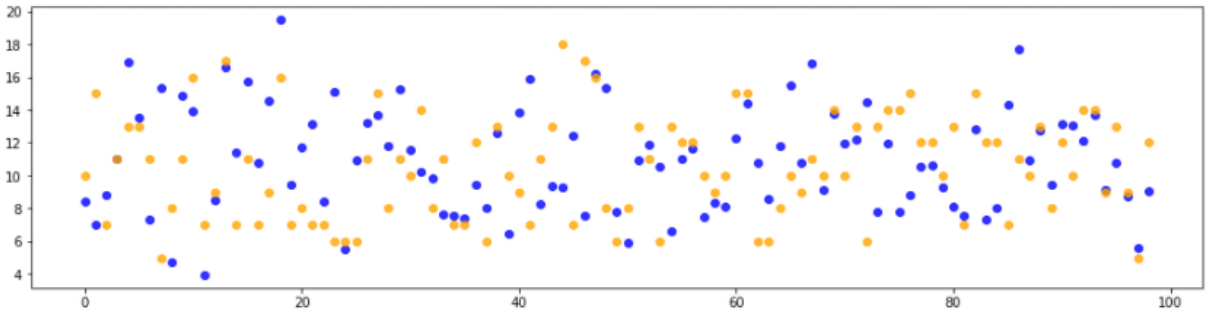
```
train loss: 3.6428396701812744
test loss: 25.823131561279297
```



雖然 test loss 變高，但是從直方圖來說變得更穩了，偏差較大的資料也變少了。



對照組



最終比較組

七、結論

1.loss 值能降低，但效益不大、成果不明顯，可能的原因為資料量不夠以及資料種類過於繁多

2.loss 值不只會逐漸降低。還會有許多有趣的變化(如大幅度擺動、完全不擺動)

3.每筆資料的 loss 值到最低點後會在附近徘徊，這個 loss 值不一定是 0

八、未來展望

前面說到，資料種類過於複雜且多會造成結果難以預測、不可靠，那為何我們不捨棄一些數據種類，只保留部份去做呢?因此之後可以朝向**"評估哪些數據最具影響力"**、**"一次最多做多少部分、掌握多少種類的結果是可靠的"**等方向研究。

另外，我認為原本**資料型別為數字**的資料種類會更好研究，因為文字的資料種類在經過 pandas.dummie 後只會留下 1 跟 0，這對數字統計來說不太友善，相反的如果有數字，歸納、找平均值、標準差等會方便得多，會得到更為客觀的結果。

九、心得

在這次的主題中，我主要學到了:

(1) Python 強大的 import 應用

(2) Tensorflow 的應用

(3) Python 印製圖表的功能

(4) 大數據處理的基本能力

(5) 將 csv 檔等資料轉換成電腦可以用來處理的資訊

這次的課程在一開始對我來說有點困難，原因在於我不知道一些專有名詞的意義(如 train loss、test loss 等)，又不知道這個專題的邏輯概念。後來在問過老師之後才有一些對它的理解（我覺得前導作業的畫圖跟數學課的不等式單元幫助我的理解很多），這次的課程也讓我了解到 python 的強大之處，可以 import 各種插件，幫助我們完成目的。

十、資料來源：黃敦紀老師的講義、tensorlow 官網

淺談日本與台灣空氣品質

作者:

208[redacted] 林[redacted]毅

老師: 黃敦紀老師

一、 專題目標

透過日本與台灣部分地區空氣品質指標之比較驗證是否日本的空氣品質狀況比台灣好

二、前言

由於許多原因，台灣一直為空氣品質差所詬病，許多人也藉此希望政府能制定政策改善空汙。的確空氣汙染是目前我們必須關注的議題，環境與經濟的權衡也是大家必須思考的問題。然而台灣的空氣相較其他國家，真的有比較不好嗎?本篇專題目標期望用大數據蒐集資料並觀察日本與台灣的空氣品質指標。

三、研究方法

透過蒐集一周內台灣新北市新莊區、苗栗縣苗栗市與高雄市仁武區以及日本東京都秋葉原、琦玉縣東松山市、北海道旭川市的 AQI 資料進行統整與分析。

四、AQI 介紹

甚麼是 AQI?為何用此指標作為空汙判斷標準?

空氣品質指標 AQI 是政府從 2016 年 12 月，開始實施的空氣品質監測標準，測量的污染項目包含六種，分別是臭氧(O3)、細懸浮微粒(PM2.5)、懸浮微粒(PM10)、一氧化碳(CO)、二氧化硫(SO2)及二氧化氮(NO2)，將不同污染物對人體健康的影響程度，分別換算出副指標值，再以當日各副指標之最大值為該測站當日之空氣品質指標值(AQI)。AQI 相較其他空氣品質判斷標準而言更為準確、廣泛且具可參考性，故本研究採用 AQI 指標進行數據分析。

六、資料獲取方法

七、實驗與結果分析

台灣

利用 json editor 以及以下由老師提供的網址
(https://data.epa.gov.tw/api/v1/aqx_p_432?limit=1000&api_key=9be7b239-557b-4

c10-9775-78cadfc555e9&format=json)蒐集資料，並手動紀錄連續一個禮拜至 excel.

日本

利用網站(https://aqicn.org/map/japan/hk/) 並使用網頁爬蟲蒐集資料並手動紀錄至 excel.

統計時間:
2021/4/16 至 2021/4/23(每日台灣時間 23:00)

以下資料順序為:
1. 日本東京都台東區(秋葉原附近) 2.日本琦玉縣東松山市五領町近隣公園 3. 日本北海道旭川市 4. 新北市新莊 5. 苗栗縣苗栗 6.高雄市仁武

統計結果:

時間/地點	4.16	4.17	4.18	4.19	4.2	4.21	4.22	4.23
東京	32	22	40	43	28	15	23	17
琦玉	3	10	6	3	12	7	12	2
北海道	16	7	12	21	11	10	13	21
新莊	48	32	28	25	26	39	32	35
苗栗	52	41	40	22	33	48	53	69
仁武	43	36	32	48	17	36	50	50
日本平均:	17	13	19	22	17	10	16	13
台灣平均:	47	36	33	31	25	41	45	51

數據分析:

1.七天的平均可以看出日本的三個地區 AQI 資料取平均在這七天均小於台灣之三個地區，雖不能一言以蔽日本的空氣品質都比台灣好，但從這六筆資料可以看出些許端倪。

2.其中有兩個特別極端的資料地區，分別是日本的琦玉縣與台灣的苗栗縣，以下將嘗試解釋為何這兩筆資料這麼明顯:

日本琦玉縣: 我認為該筆資料 AQI 會如此低的原因在於:
(1).觀測地點位於公園內，公園的環境有助於提升空氣品質
(2).東松山市的人口僅有 9 萬，相較於苗栗市的 8.7 萬相差不遠，但土地面積則是苗栗市的兩倍，較低的人口密度較不會造成空氣汙染
(3)東松山市為琦玉縣相對以觀光為主要收入的市，為維護觀光品質，政府有特別管控空氣汙染係數

台灣苗栗市
(1).與日本琦玉縣相較而言，人口密度高較容易造成空氣汙染
(2).觀測站位於苗栗市的長春石化工業區附近，故工廠排放的廢氣容易影響到空氣品質

雖然時間的深度與參考地區的廣度還不夠作為有決定性、批判性的資訊，但在找資料的過程中也讓我學會了解決問題以及對獲取知識的樂趣，更重要的是去發想一個主題並實踐它，這也是相對我欠缺的能力，感謝老師給這個機會可以讓我精進。

八、總結

在這次的主題內，我學到了:

(1) 使用 JsonEditor 將資料以大小分門別類

(2) 利用 API 網站蒐集資料並統計

(3) 使用者經驗的基礎概念(透過對話讓程式便更有趣)

這次的 API 資料抓取，我相信它有許多可以應用的情況。這是我第一次從頭到尾由自己

打完的專題，雖然相較前面兩個較為簡單，但還是給我偌大的成就感。

九、程式碼(附在本文最下方):

```
import requests
from bs4 import BeautifulSoup
print('資料接收中...請稍等')
taiwan = requests.get(
 "https://data.epa.gov.tw/api/v1/aqx_p_432?limit=1000&api_key=9be
7b239-557b-4c10-9775-78cadfc555e9&format=json"
 ).json()

gangster = taiwan["records"][6]["Status"] #新莊狀態
AQIgangster = taiwan["records"][6]["AQI"] #新莊

countryside = taiwan["records"][25]["Status"] #苗栗狀態
AQIcountryside = taiwan["records"][25]["AQI"] #苗栗

tallbear = taiwan["records"][47]["Status"] #高雄仁武狀態
AQItallbear = taiwan["records"][47]["AQI"] #高雄仁武


res =
requests.get('https://aqicn.org/city/japan/taitoku/meijidoriozeki
yokocho/hk/')
soup = BeautifulSoup(res.text, 'html.parser')
japan1 = soup.find("div" , class_ = 'aqivalue'   ) #日本東京都台東區(秋葉原附近)


res =
requests.get('https://aqicn.org/city/japan/higashimatsuyamashi/hi
gashimatsuyama/hk/')
soup = BeautifulSoup(res.text, 'html.parser')
```

```
japan2 = soup.find("div" , class_ = 'aqivalue'   ) #日本琦玉縣東松山市五領町近隣公園


res =
requests.get('https://aqicn.org/city/japan/asahikawashi/toko/hk')
soup = BeautifulSoup(res.text, 'html.parser')
japan3 = soup.find("div" , class_ = 'aqivalue'   ) #日本北海道北門旭川市


pre = input('請問你知道甚麼是 AQI(空氣品質指標)嗎? (請回答 yes/no)')
if(pre == 'no'):
 print('空氣品質指標為依據監測資料將當日空氣中臭氧(O3)、細懸浮微粒(PM2.5)、懸浮微粒(PM10)、一氧化碳(CO)、二氧化硫(SO2)及二氧化氮(NO2)濃度等數值，以其對人體健康的影響程度，分別換算出不同污染物之副指標值，再以當日各副指標之最大值為該測站當日之空氣品質指標值(AQI)。')
 print('  ')
print('很棒!你知道甚麼是 AQI 了，接著讓我們繼續下去吧!')
a = input('請問要列印各區 AQI 嗎? (請回答 yes/no)')
b = input('請問要列印六區比較嗎? (請回答 yes/no)')
if(a == 'yes'):
 print("日本東京都台東區(秋葉原附近)")
 print("AQI:",japan1.text)

 print("日本琦玉縣東松山市五領町近隣公園")
 print("AQI:",japan2.text)

 print("日本北海道北門旭川市")
 print("AQI:",japan3.text)

 print('新北市新莊區')
 #print(gangster)
 print('AQI:',AQIgangster)

 print('苗栗縣苗栗市')
 #print(countryside)
 print('AQI:',AQIcountryside)

 print('高雄市仁武區')
```

```
 #print(tallbear)
 print('AQI:',AQItallbear)
#以下台灣

allAQI = list()

allAQI.append(japan1.text )
allAQI.append(japan2.text )
allAQI.append(japan3.text )
allAQI.append(AQIgangster)
allAQI.append(AQIcountryside)
allAQI.append(AQItallbear)

if( a == b and a == 'yes'):
 print('-----------------------------')

if( b == 'yes'):

 print('空氣品質最差的是')
 if(max(allAQI) == japan1.text) :
   print('日本東京都台東區(秋葉原附近)')
 if(max(allAQI) == japan2.text) :
   print('日本琦玉縣東松山市五領町近隣公園')
 if(max(allAQI) == japan3.text) :
   print('日本北海道北門旭川市')
 if(max(allAQI) == AQIgangster) :
   print('新北市新莊區')
 if(max(allAQI) == AQIcountryside) :
   print('苗栗縣苗栗市')
 if(max(allAQI) == AQItallbear) :
   print('高雄市仁武區')

 print('空氣品質最好的是')
 if(min(allAQI) == japan1.text) :
   print('日本東京都台東區(秋葉原附近)')
 if(min(allAQI) == japan2.text) :
   print('日本琦玉縣東松山市五領町近隣公園')
 if(min(allAQI) == japan3.text) :
```

```
   print('日本北海道北門旭川市')
 if(min(allAQI) == AQIgangster) :
   print('新北市新莊區')
 if(min(allAQI) == AQIcountryside) :
   print('苗栗縣苗栗市')
 if(min(allAQI) == AQItallbear) :
   print('高雄市仁武區')
 twAQI = ((int(AQIgangster) + int(AQIcountryside )+ int(AQItallbear))
/3 )
 jpAQI = ((int(japan1.text) + int(japan2.text) + int(japan3.text))
/ 3)
print('台灣 AQI 平均:',twAQI)
print('日本 AQI 平均:',jpAQI)
if a == 'no' and b == 'no':
    print('ummm')
```

課程內容補充

密碼學：去猜測一段上鎖後的密碼的原句。

1.

Oat pvux'b ou mftyxe?

Huqpwsu eubbyxe bauvu ys apfm bau mwx.

Y opxb bi fyku dt fymu, xib vuqivg yb.

Why aren't we flying?

because getting there is half the fun.

I want to live my life, not record it.

1.猜 Y = I

2.因 pvux'b 猜

b = t

3.設ti為 to 猜 I = o

4.第一句為問句 問句通常5W 開頭 所以設 O = W

5.i ____ to 設 I want to 則 p = a, x = n

6.w_為名詞 因此猜 u = e

7.如 ou = we 則猜前面為 aren’t 則 v = r

8. w_ _ aren’t we 且為三字母 問句為 Why a = h, t = y

9._i_e 且為動詞,猜 live f = l ,k = v

10.若_e_ in_ 設為形容詞或動詞則尾端為 ing e = g

11. hal_ 猜 half m = f

12.f_n 猜 fun w = u

13.放句首 且為_e_ause 猜 because h = b , q = c

14.l_fe 且前面有活(live)猜 life y = I

15.recor_ 猜 record g = d

16.經 google 搜尋

I want to live _y life, not record it.

得知此句為名言佳句「我想要生活，不是紀錄生活。」

– 賈桂琳·甘迺迪(美國第一夫人)

2.Enigma 破譯機(Vernam Cipher)原理應用

祕文一11 37 18 22 29 6 21 10 32 22 26 28 20 18 27 25 29 14 29 21 22 36 20 999

祕文二 3 43 30 27 14 10 25 8 33 36 14 15 44 32 30 9 29 16 31 24 22 32 22 999

祕文三 4 34 11 17 24 10 25 22 32 32 5 7 25 31 30 25 28 12 36 21 32 34 40 999

提示:john, crazy

由 crazy 帶進去 ：

1.2 etdzu asonl 1.3 orked puayw 2.1 x 2.3 croyu brqyu 3.1 cscsy jwfvu 3.2 gmdpw vsajz mhazk

1.貌似只有 1.3 orked 1.2 asonl 比較像英文 猜 為 worked

2. 3代1 worked-----scrazy , crazy 為形容詞 前面通堂為 be 動詞 則設為 is crazy 得 mworked 感覺不合

3.突然想到有過去式 代 1 3 wascrazy 得 re worked

4.發現題目有 john 代 john was crazy 試試看 得 Claire worked

5.人名+worked 介係詞有 as for at 代入後 for 最合 得 john was crazy abo

6.abo 猜介係詞 about 得 Claire worked for us

7. Claire worked for us 後面通常加介係詞或副詞 先猜 as , at as 代進去較合得 john was crazy about cl

8.john was crazy about cl　猜為 claire 得 Claire worked for us as aspy 分成 Claire worked for us as a spy
9.將第一句帶入 第二句為:butshewasolnypretending 分解為 but she was only pretending.

已經得出第一句為:John was crazy about Claire.

第二句為:But she was only pretending.

第三句為:Claire worked for us as a spy.