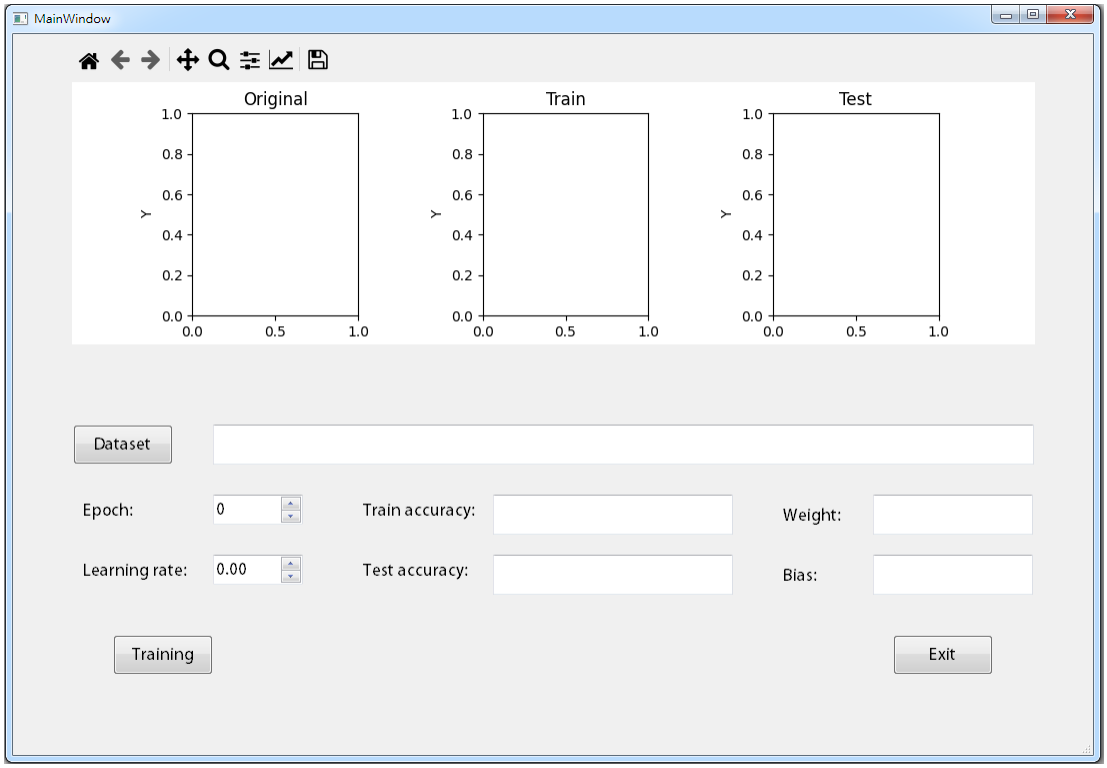
類神經網路作業一 - 設計感知機類神經網路

一、程式執行說明 (GUI 功能說明)



1. 上面三張圖 Original、Train、Test 會畫出原本的分類、訓練資料的分類結果、測試資料的分類結果
2. Dataset Button 可以選擇 txt 檔案輸入，路徑會顯示在旁邊
3. Epoch、Learning rate 旁邊的 spinbox 可以輸入想要的 epoch 以及學習率
4. Train accuracy 及 Test accuracy 旁邊的文字框會顯示訓練準確率與測試準確率
5. Weight 和 Bias 旁的文字框會顯示權重(=鍵結值)與偏置(=閥值/閾值)
6. 選完檔案後，按 Training 開始訓練，沒有選檔案程式會關閉
7. Exit 或是右上角 X 關閉程式
8. 圖形上面的功能欄可以對各個子圖作操作，分別是初始化、上一步、下一步、移動畫面、放大、更改圖的邊界與間距、編輯子圖、儲存畫出的圖形

二、程式碼簡介

1. UI.py 為用 Qt designer 設計出來的介面轉成的程式碼
2. Mplwidget.py 則是將 matplotlib 嵌入介面所需的程式碼
3. Perceptron_controller.py 則是主程式，包含拆分資料、找出感知機的分類線、畫圖、計算準確率等等

A. __init__ 用於初始化及設定 UI

```
class MatplotlibWidget(QtWidgets.QMainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.setup_control()

        self.filename = ""
        self.points = np.empty([0, 2], float)
        self.pclass = np.array([], int)
        self.allclass = np.array([], int)
        self.pred = np.array([], int)
        self.epoch = 0
        self.learning_rate = 0
        self.train_accuracy = 0
        self.test_accuracy = 0
```

B. reset 是每次重新畫圖時重置 attribute，像是重置訓練準確率、測試準確率等等

```
def reset(self):
    self.points = np.empty([0, 2], float)
    self.pclass = np.array([], int)
    self.allclass = np.array([], int)
    self.pred = np.array([], int)
    self.epoch = 0
    self.learning_rate = 0
    self.train_accuracy = 0
    self.test_accuracy = 0
```

C. setup_control 則是將 button 與功能連結在一起，當按下某個 button 則會做某項函式

```
def setup_control(self):  # botton 連接加在這裡
    self.ui.dataset_button.clicked.connect(self.open_file)
    self.ui.exit_button.clicked.connect(self.exit)
    self.ui.training_button.clicked.connect(self.train_control)
```

D. open_file 是開啟檔案的功能，當按下 botton 後就會開啟選擇檔案的介面，選完會設定檔案路徑，並顯示路徑在文字框裡

```
def open_file(self):
    self.filename, filetype = QFileDialog.getOpenFileName(self,
                                                          "Open file",
                                                          "./")  # start path
    self.ui.show_file_path.setText(self.filename)
    font = QtGui.QFont()
    font.setFamily("Adobe 繁黑體 Std B")
    font.setPointSize(12)
    font.setBold(True)
    font.setWeight(75)
    self.ui.show_file_path.setFont(font)
```

E. start_train 負責將輸入資料轉為 nparray

```
def start_train(self):

    with open(self.filename) as f:
        lines = f.readlines()
        for line in lines:
            x, y, d = line.split(" ")
            self.pclass = np.append(self.pclass, int(d[0]))
            self.points = np.append(self.points, np.array([[x, y]], dtype="float64"), axis=0)
            if int(d[0]) not in self.allclass:
                self.allclass = np.append(self.allclass, int(d[0]))
    np.sort(self.allclass)
```

F. split_data 負責隨機分割訓練資料與測試資料，具體實現方式是用陣列存放打亂的 index

```
def split_data(self):
    shuffled_indices = np.random.permutation(len(self.points))
    train_data_size = int(len(self.points) * (2 / 3))

    self.train_indices = shuffled_indices[:train_data_size]
    self.test_indices = shuffled_indices[train_data_size:]
```

G. train_predict 負責訓練感知機，每一次的 epoch 都會將所有訓練資料跑過一輪，並修正感知機權重(鍵結值)與偏置(閥值)

預測分類：計算 $w^T x + b$ 是否>0，也就是計算輸入的點代入感知機的線會位於哪方

修正公式：

$$w_i \leftarrow w_i + \eta(t - y)x_i$$

$$b \leftarrow b + \eta(t - y)$$

其中 w 為權重，η 為學習率，t 為實際的分類，y 為預測的分類，x 為輸入，b 為偏置

```
def train_predict(self):
    unit_step = lambda x: self.allclass[0] if x < 0 else self.allclass[1]  # 活化函數
    w = np.random.rand(len(self.points[0]))
    bias = 0

    for i in range(self.epoch):
        for j in self.train_indices:
            input_i = self.points[j]
            label = self.pclass[j]
            result = input_i * w + bias
            result = float(sum(result))
            y_pred = float(unit_step(result))
            w = w + self.learning_rate * \
                (label - y_pred) * np.array(input_i)  # 更新權重
            bias = bias + self.learning_rate * (label - y_pred)  # 更新 bias

    return w, bias
```

H. train_predict 用找到的感知機分類線再次使用訓練資料判斷屬於哪一類，並且也使用測試資料判斷屬於哪一類

```
def test_predict(self, w, bias):  # 預測函數
    """Return class label after unit step"""
    unit_step = lambda x: self.allclass[0] if x < 0 else self.allclass[1]  # 活化函數

    for i in self.train_indices:
        input_i = self.points[i]
        result = input_i * w + bias
        result = float(sum(result))
        y_pred = float(unit_step(result))
        self.pred = np.append(self.pred, y_pred)

    for i in self.test_indices:
        input_i = self.points[i]
        result = input_i * w + bias
        result = float(sum(result))
        y_pred = float(unit_step(result))
        self.pred = np.append(self.pred, y_pred)

    return
```

I. train_plot 畫出將訓練資料用感知機分類出的結果，並計算訓練準確率

```
def train_plot(self, w, bias):
    self.ui.widget.canvas.train.cla()
    for i in range(len(self.train_indices)):
        j = self.train_indices[i]
        if self.pred[i] == 1:
            self.ui.widget.canvas.train.scatter(self.points[j, 0], self.points[j, 1], s=5, color='r')
        else:
            self.ui.widget.canvas.train.scatter(self.points[j, 0], self.points[j, 1], s=5, color='b')

        if self.pred[i] == self.pclass[j]:
            self.train_accuracy += 1

    self.train_accuracy = self.train_accuracy / len(self.train_indices)

    x1 = -bias / w[0]

    self.ui.widget.canvas.train.axline([x1, 0], slope=-w[0] / w[1], lw=3, color='k')
    self.ui.widget.canvas.train.axis(xmin=min(self.points[:, 0]) - 1, xmax=max(self.points[:, 0]) + 1)  # 設定X軸顯示範圍
    self.ui.widget.canvas.train.axis(ymin=min(self.points[:, 1]) - 1, ymax=max(self.points[:, 1]) + 1)  # 設定Y軸顯示範圍
    self.ui.widget.canvas.train.set_xlabel("X")
    self.ui.widget.canvas.train.set_ylabel("Y")
    self.ui.widget.canvas.train.set_title("Train")
```

J. test_plot 畫出將測試資料用感知機分類出的結果，並計算測試準確率

```
def test_plot(self, w, bias):
    self.ui.widget.canvas.test.cla()
    for i in range(len(self.test_indices)):
        j = self.test_indices[i]
        if self.pred[len(self.train_indices) + i] == 1:
            self.ui.widget.canvas.test.scatter(self.points[j, 0], self.points[j, 1], s=5, color='r')
        else:
            self.ui.widget.canvas.test.scatter(self.points[j, 0], self.points[j, 1], s=5, color='b')

        if self.pred[len(self.train_indices) + i] == self.pclass[j]:
            self.test_accuracy += 1

    self.test_accuracy = self.test_accuracy / len(self.test_indices)

    x1 = -bias / w[0]

    self.ui.widget.canvas.test.axline([x1, 0], slope=-w[0] / w[1], lw=3, color='k')
    self.ui.widget.canvas.test.axis(xmin=min(self.points[:, 0]) - 1, xmax=max(self.points[:, 0]) + 1)  # 設定X軸顯示範圍
    self.ui.widget.canvas.test.axis(ymin=min(self.points[:, 1]) - 1, ymax=max(self.points[:, 1]) + 1)  # 設定Y軸顯示範圍
    self.ui.widget.canvas.test.set_xlabel("X")
    self.ui.widget.canvas.test.set_ylabel("Y")
    self.ui.widget.canvas.test.set_title("Test")
```

K. original_plot 畫出原本資料的分群，並顯示感知機的分類線

```
def original_plot(self, w, bias):
    self.ui.widget.canvas.original.cla()

    for i in range(len(self.points)):
        if self.pclass[i] == 1:
            self.ui.widget.canvas.original.scatter(self.points[i, 0], self.points[i, 1], s=5, color='r')
        else:
            self.ui.widget.canvas.original.scatter(self.points[i, 0], self.points[i, 1], s=5, color='b')

    x1 = -bias / w[0]

    self.ui.widget.canvas.original.axline([x1, 0], slope=-w[0] / w[1], lw=3, color='k')
    self.ui.widget.canvas.original.axis(xmin=min(self.points[:, 0]) - 1, xmax=max(self.points[:, 0]) + 1)  # 設定X軸顯示範圍
    self.ui.widget.canvas.original.axis(ymin=min(self.points[:, 1]) - 1, ymax=max(self.points[:, 1]) + 1)  # 設定Y軸顯示範圍
    self.ui.widget.canvas.original.set_xlabel("X")
    self.ui.widget.canvas.original.set_ylabel("Y")
    self.ui.widget.canvas.original.set_title("Original")
    self.ui.widget.canvas.figure.subplots_adjust(wspace=0.75)
```

L. train_control 則是按下 train 後會做的動作，是最主要的地方，他會依序作 attribute 重置、取得輸入的 epoch 及學習率、將輸入資料轉為 nparray、分割資料、訓練並修正感知機權重與偏置、用感知機預測訓練資料及測試資料分類、畫圖並計算準確率、顯示準確率和權重及偏置

```
def train_control(self):
    self.reset()
    self.epoch = self.ui.epoch_spinbox.value()
    self.learning_rate = self.ui.learning_rate_spinbox.value()

    self.start_train()
    self.split_data()
    w, bias = p.train_predict()
    self.test_predict(w, bias)
    bias *=2
    self.train_plot(w, bias)
    self.test_plot(w, bias)
    self.original_plot(w, bias)
    self.ui.widget.canvas.draw()

    self.ui.show_train_accuracy.setText(str(p.train_accuracy))
    font = QtGui.QFont()
    font.setFamily("Adobe 繁黑體 Std B")
    font.setPointSize(12)
    font.setBold(True)
    font.setWeight(75)
    self.ui.show_train_accuracy.setFont(font)

    self.ui.show_test_accuracy.setText(str(p.test_accuracy))
    font = QtGui.QFont()
    font.setFamily("Adobe 繁黑體 Std B")
    font.setPointSize(12)
    font.setBold(True)
    font.setWeight(75)
    self.ui.show_test_accuracy.setFont(font)

    self.ui.show_weight.setText(str(w))
    font = QtGui.QFont()
    font.setFamily("Adobe 繁黑體 Std B")
    font.setPointSize(12)
    font.setBold(True)
    font.setWeight(75)
    self.ui.show_weight.setFont(font)

    self.ui.show_bias.setText(str(bias))
    font = QtGui.QFont()
    font.setFamily("Adobe 繁黑體 Std B")
    font.setPointSize(12)
    font.setBold(True)
    font.setWeight(75)
    self.ui.show_bias.setFont(font)
```

M. exit 負責按下 Exit 按鈕後會關閉程式的功能

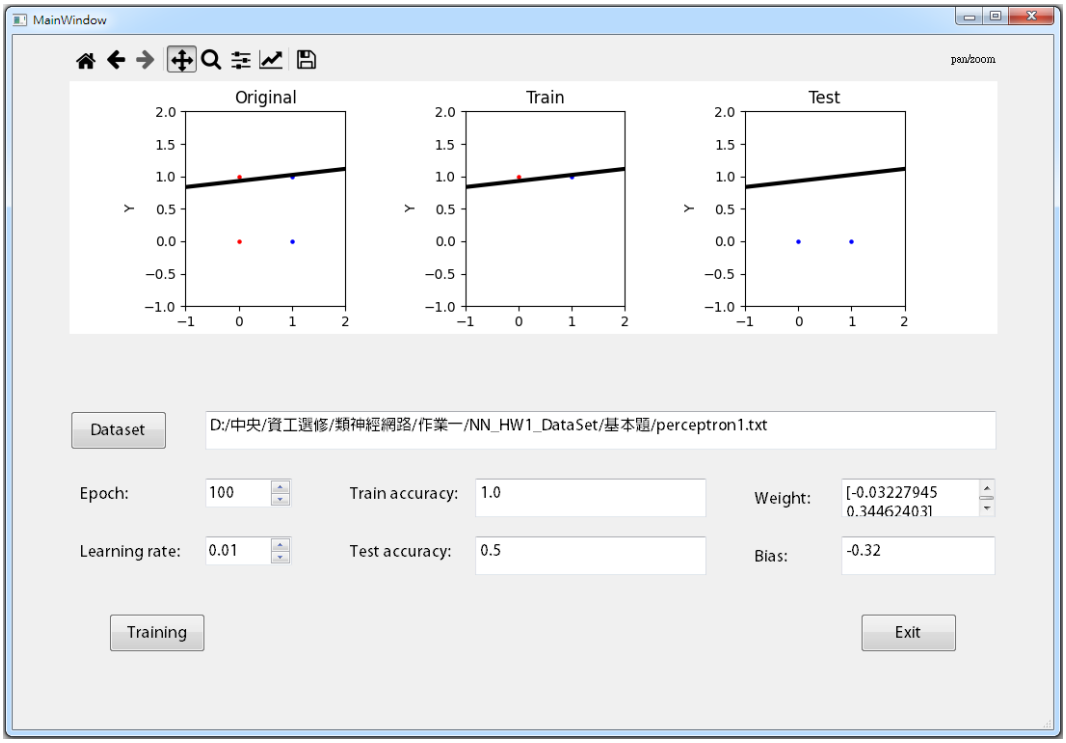
```
def exit(self):
    app.quit()
```

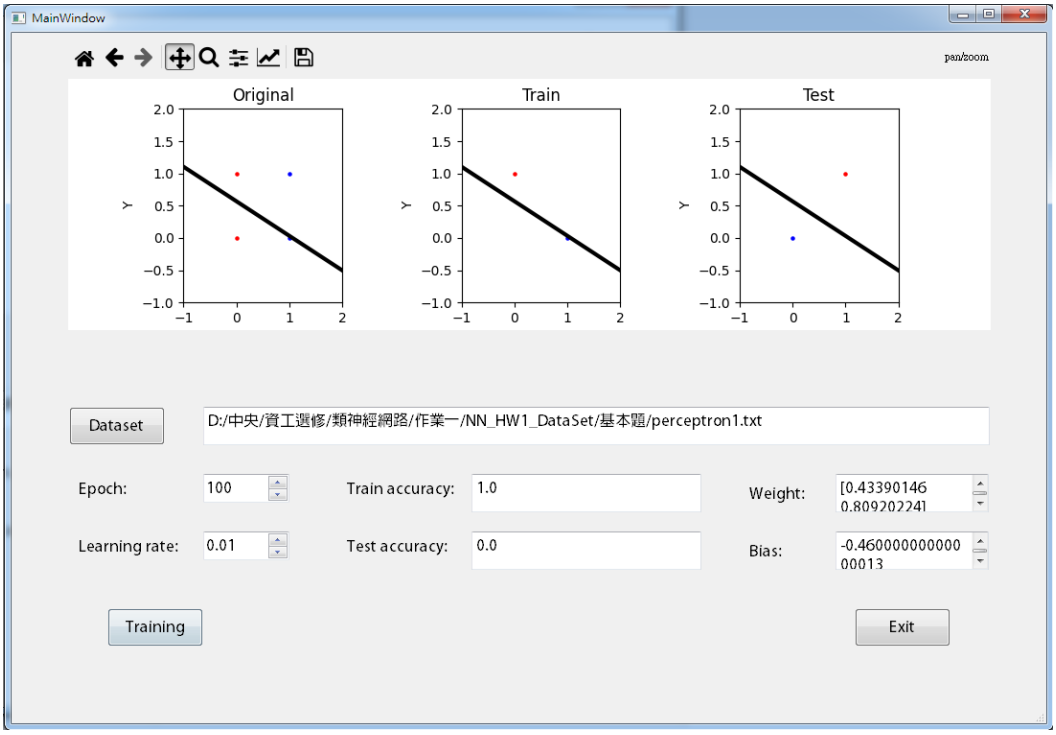
N. 啟動 exe 時，會執行 perceptron_controller 的 main 的部分，開啟應用程式，建立視窗並顯示等等

```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    p = MatplotlibWidget()
    p.show()
    sys.exit(app.exec())
```

三、實驗結果截圖、說明及分析

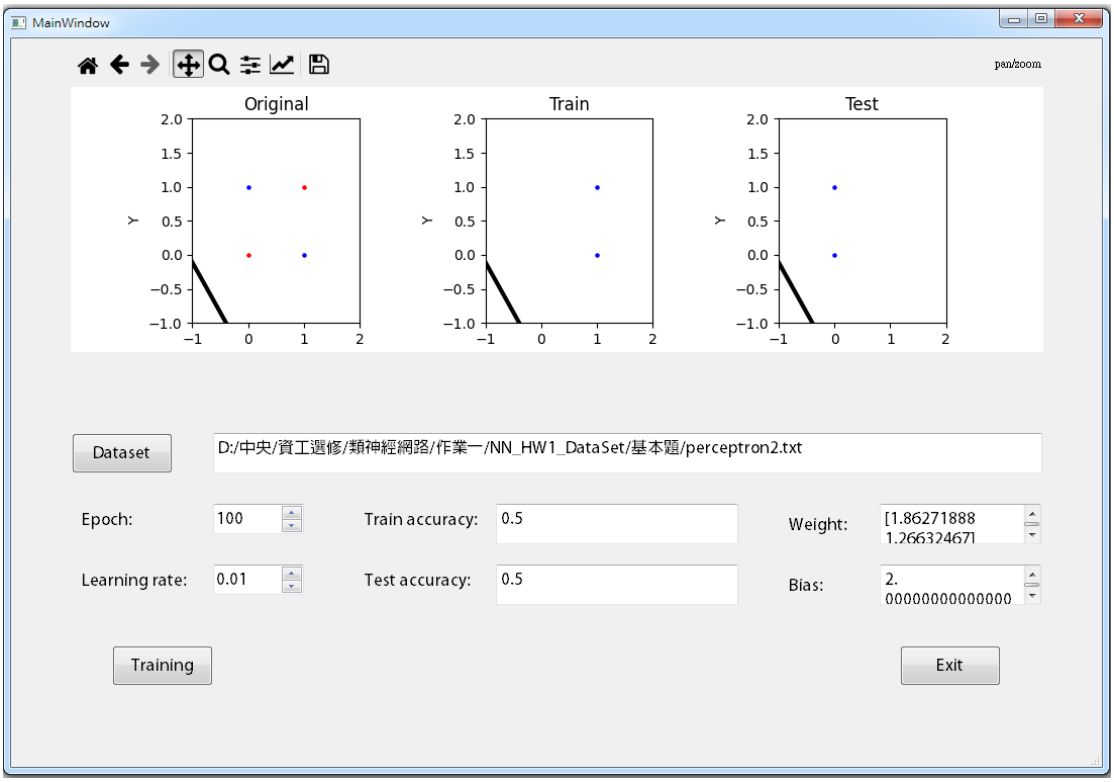
1. perceptron1

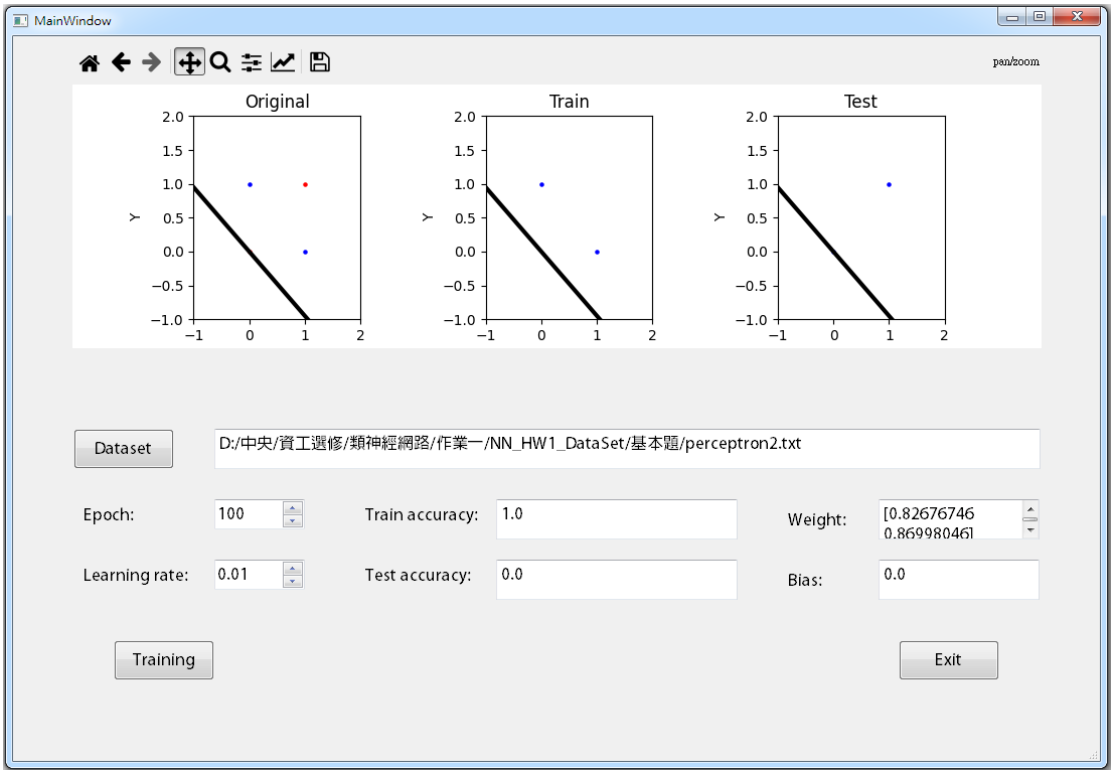




由於點的樣本數少又拆分成訓練、測試資料的關係，測試準確率常常會很低，不論 epoch 和學習率是多少，因為訓練資料只有兩個點可以分類

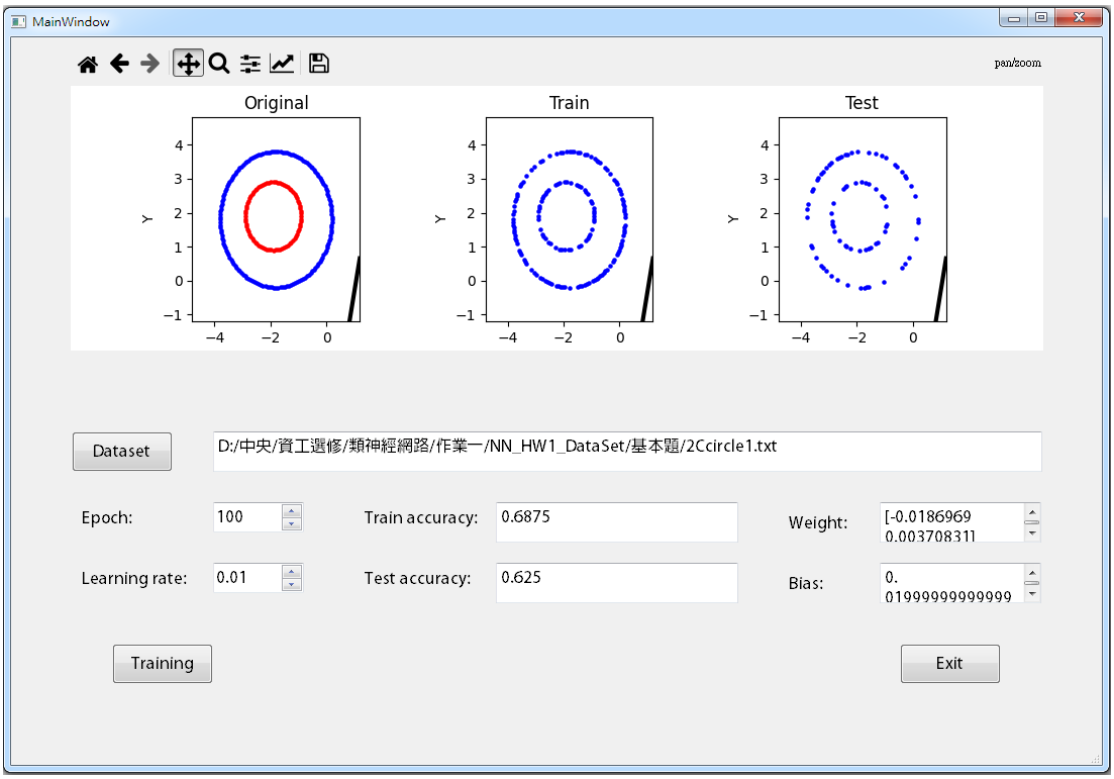
2. perceptron2

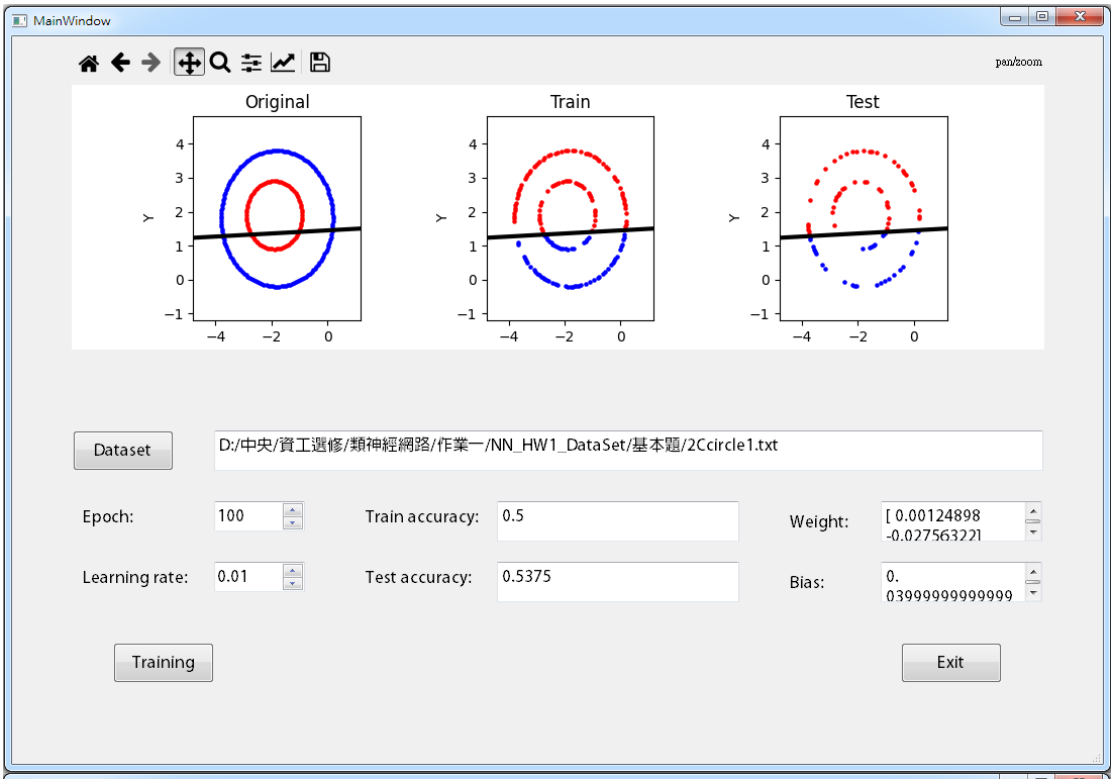


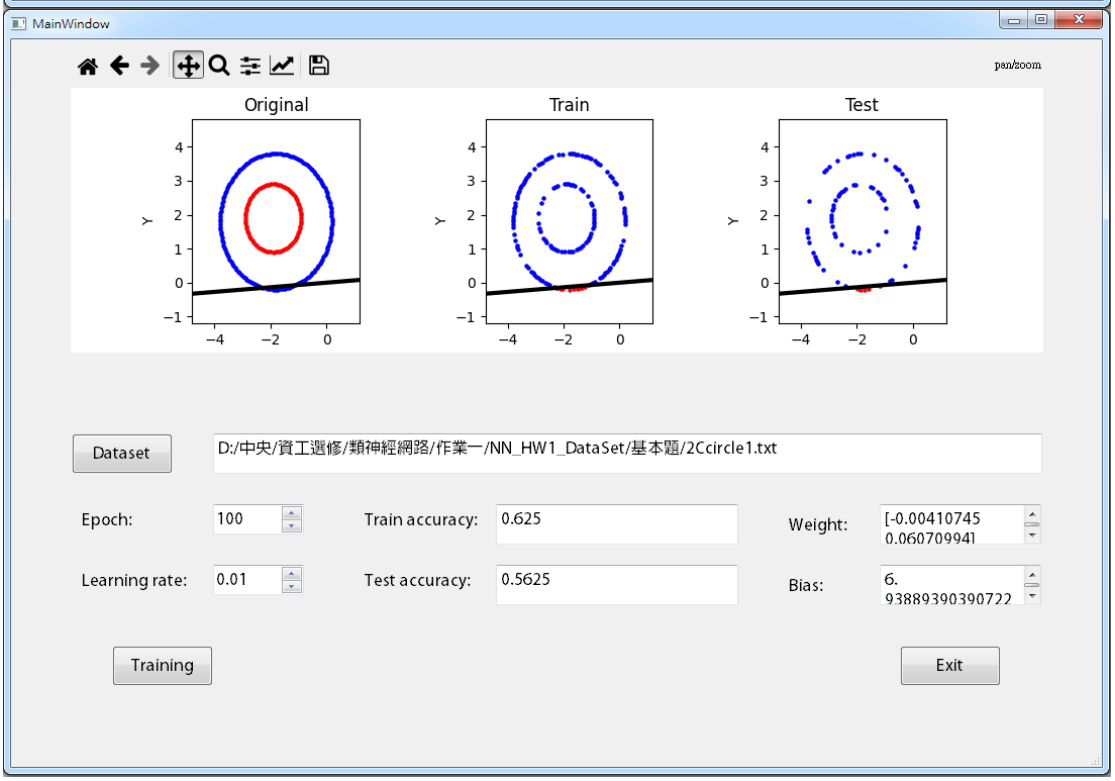


同上 perceptron1，測試準確率常常會很低，而且因為此為非線性可分的圖，準確率不太可能達 100%

3. 2Ccircle1

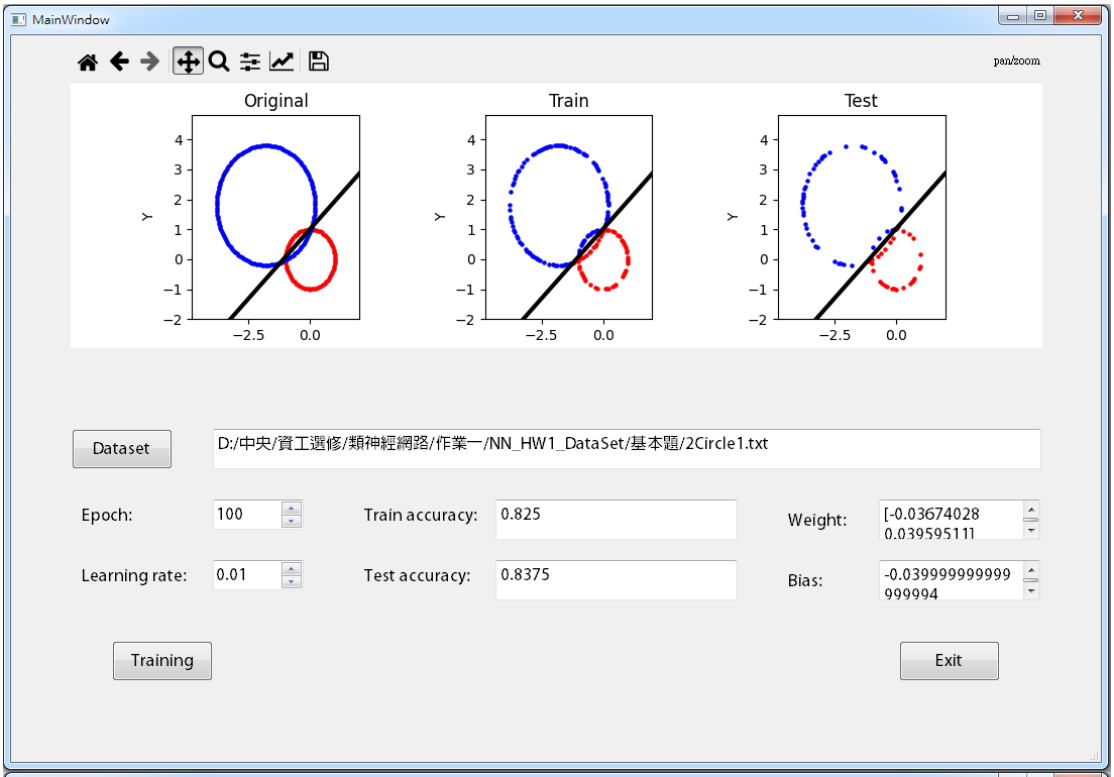


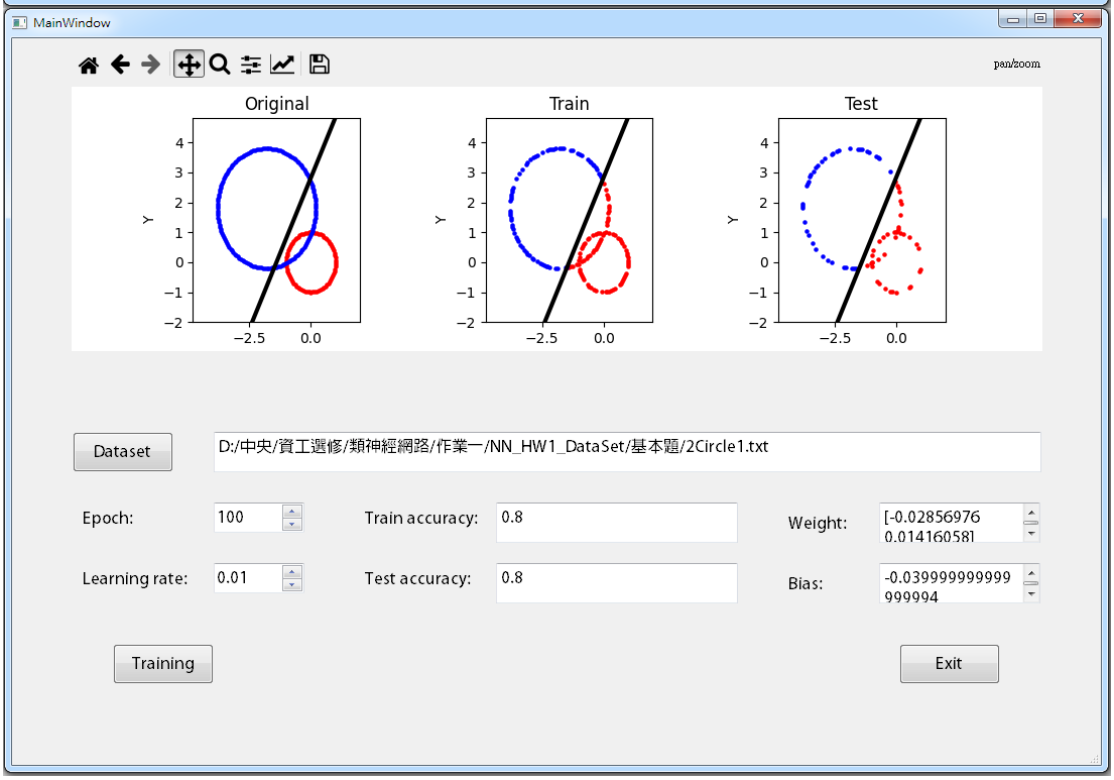




由於此為非線性可分的圖，因此準確率不高。而且感知機的線沒有規律，像是用猜的一樣。

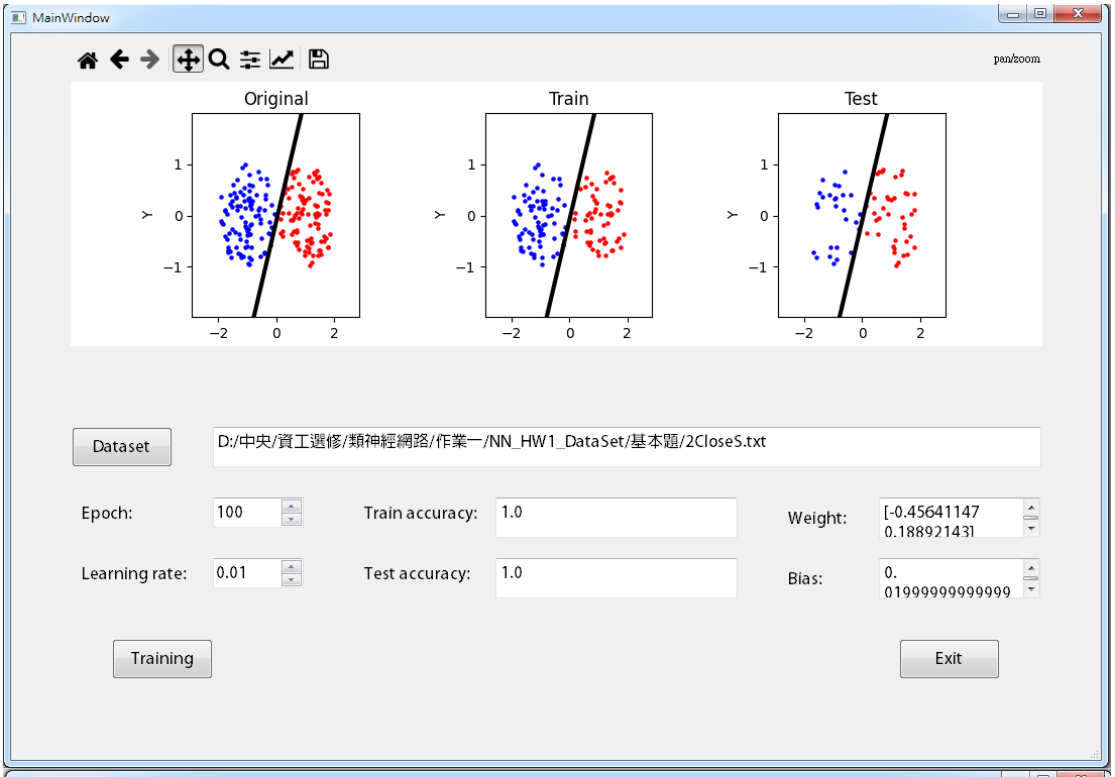
4. 2Circle1

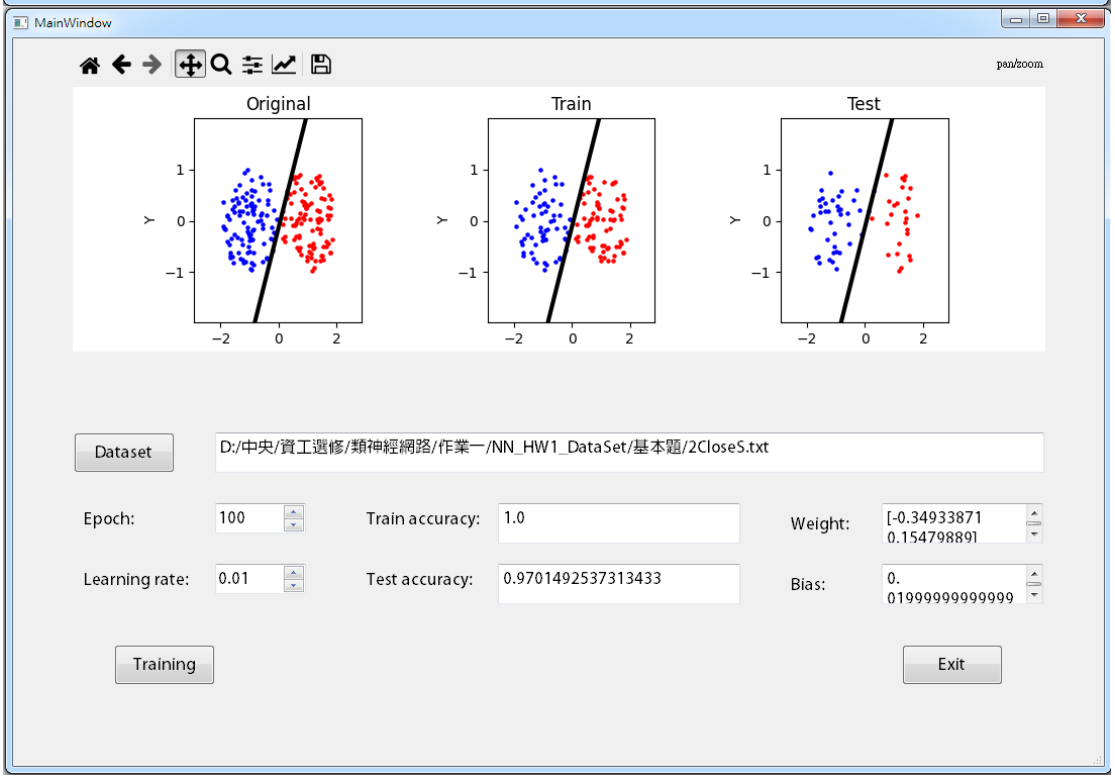


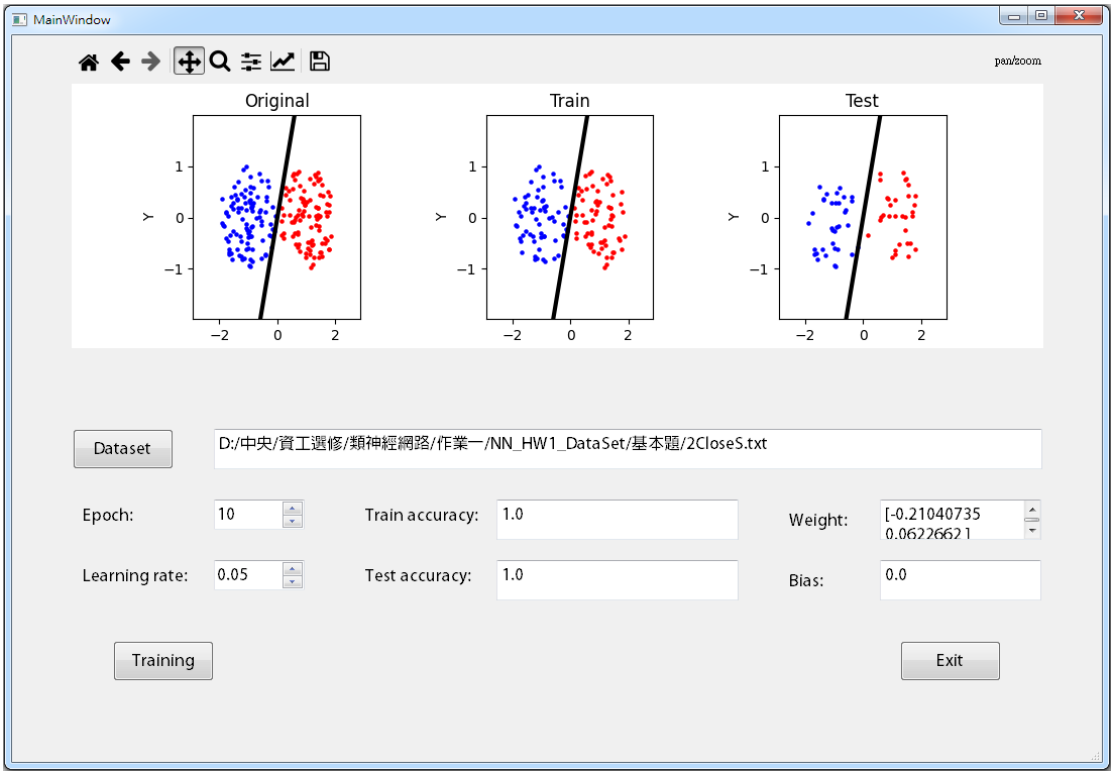


相比 2Ccircle1，2Circle1 的訓練準確率和測試準確率較高。但由於兩群形成圓形的點有交集，所以還是無法完美的將他們分開來

5. 2CloseS

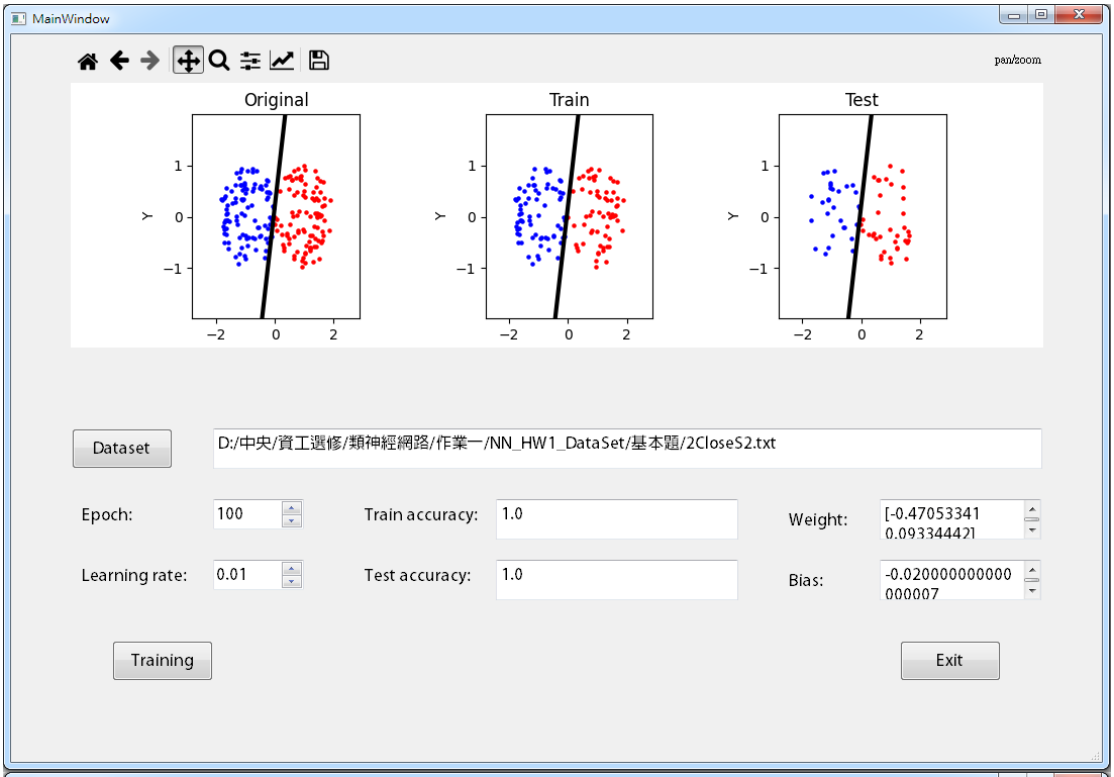

MainWindow
pan/zoom
Original
Train
Test
Dataset
D:/中央/資工選修/類神經網路/作業一/NN_HW1_DataSet/基本題/2CloseS.txt
Epoch:
100
Train accuracy:
1.0
Weight:
[-0.45641147
0.188921431
Learning rate:
0.01
Test accuracy:
1.0
Bias:
0.
01999999999999
Training
Exit

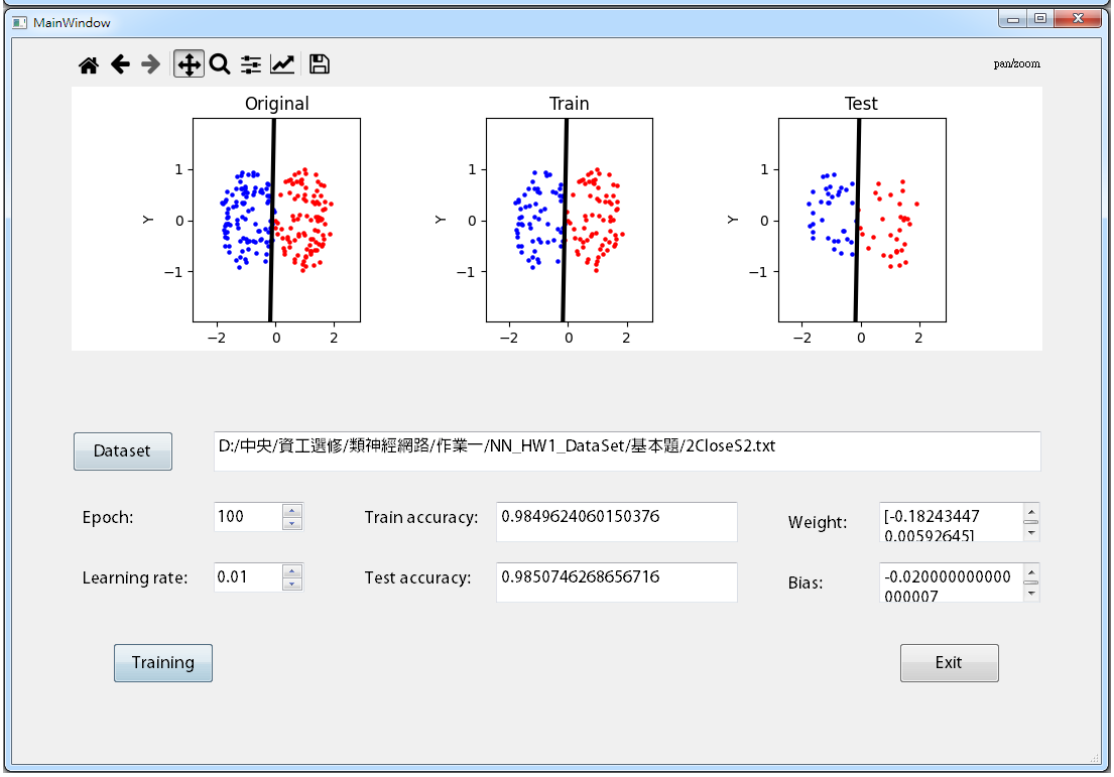

MainWindow
pan/zoom
Original
Train
Test
Dataset
D:/中央/資工選修/類神經網路/作業一/NN_HW1_DataSet/基本題/2CloseS.txt
Epoch:
100
Train accuracy:
1.0
Weight:
[-0.34933871
0.154798891
Learning rate:
0.01
Test accuracy:
0.9701492537313433
Bias:
0.
01999999999999
Training
Exit

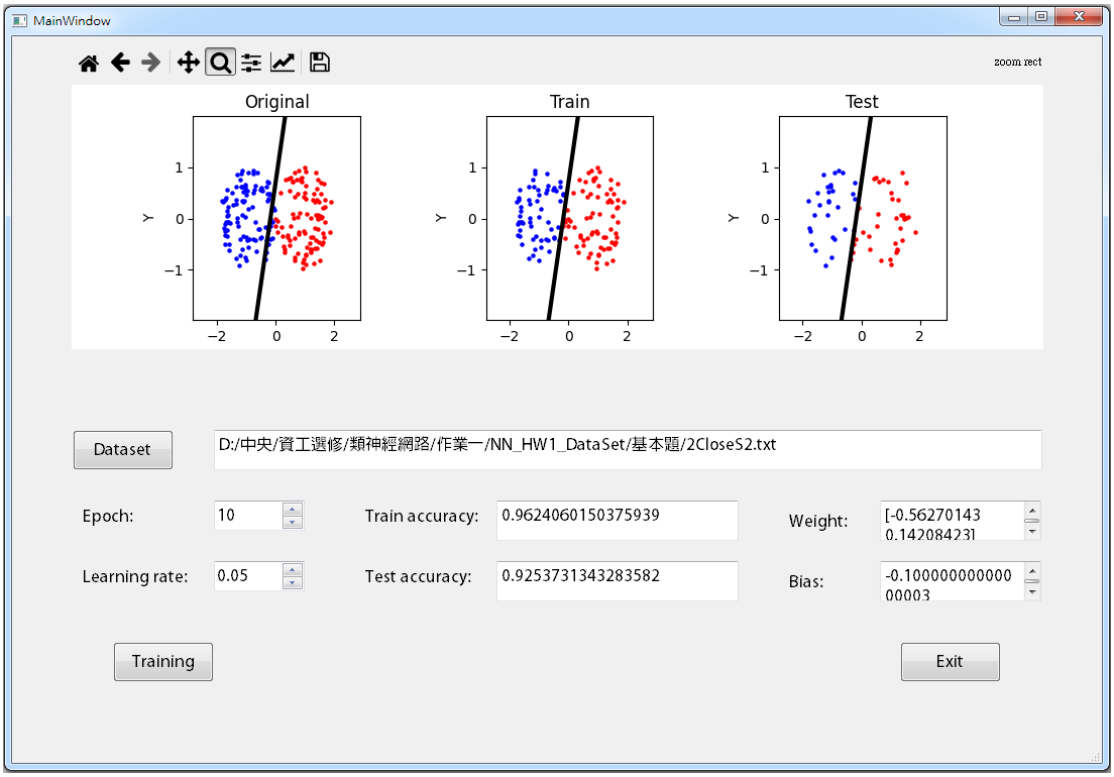


2CloseS 的點是線性可分的，所以基本上可以完美的分類，訓練準確率跟測試準確率基本都是 1，就算 epoch 設很小、學習率稍微設大一點也不太會影響，不過有時候還是會誤判幾個點

6. 2CloseS2

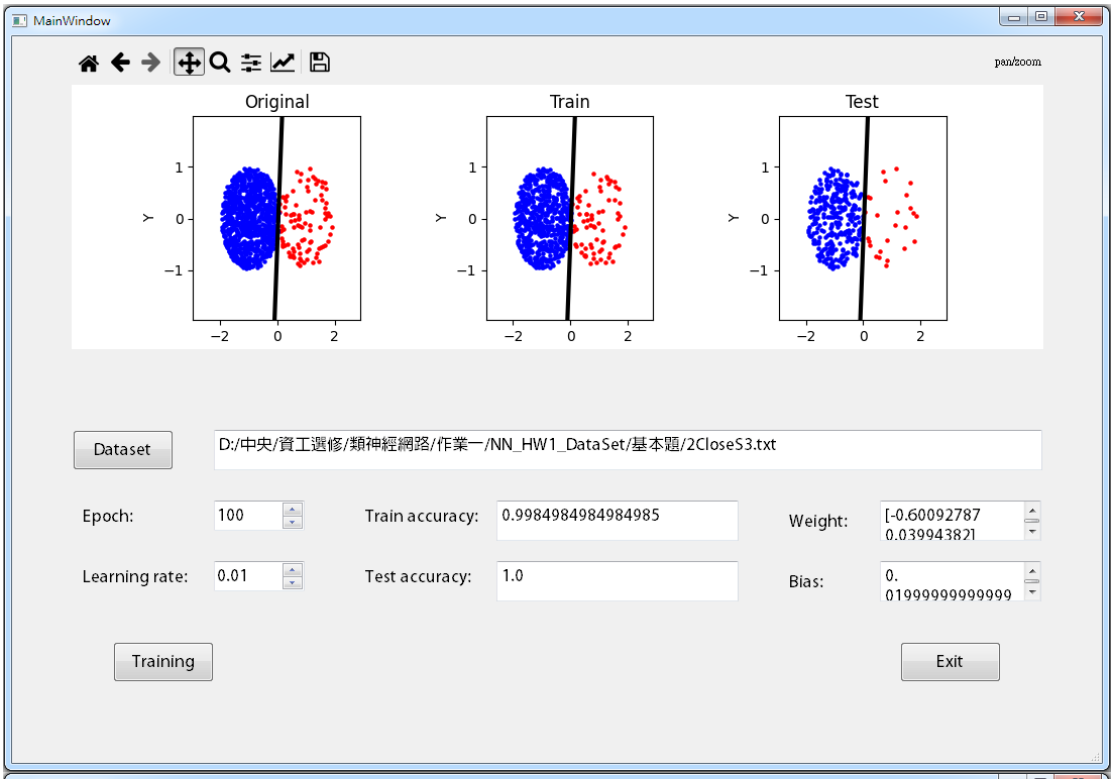

MainWindow
pan/zoom
Original
Train
Test
Dataset
D:/中央/資工選修/類神經網路/作業一/NN_HW1_DataSet/基本題/2CloseS2.txt
Epoch:
100
Train accuracy:
1.0
Weight:
[-0.47053341
0.093344421
Learning rate:
0.01
Test accuracy:
1.0
Bias:
-0.020000000000
000007
Training
Exit

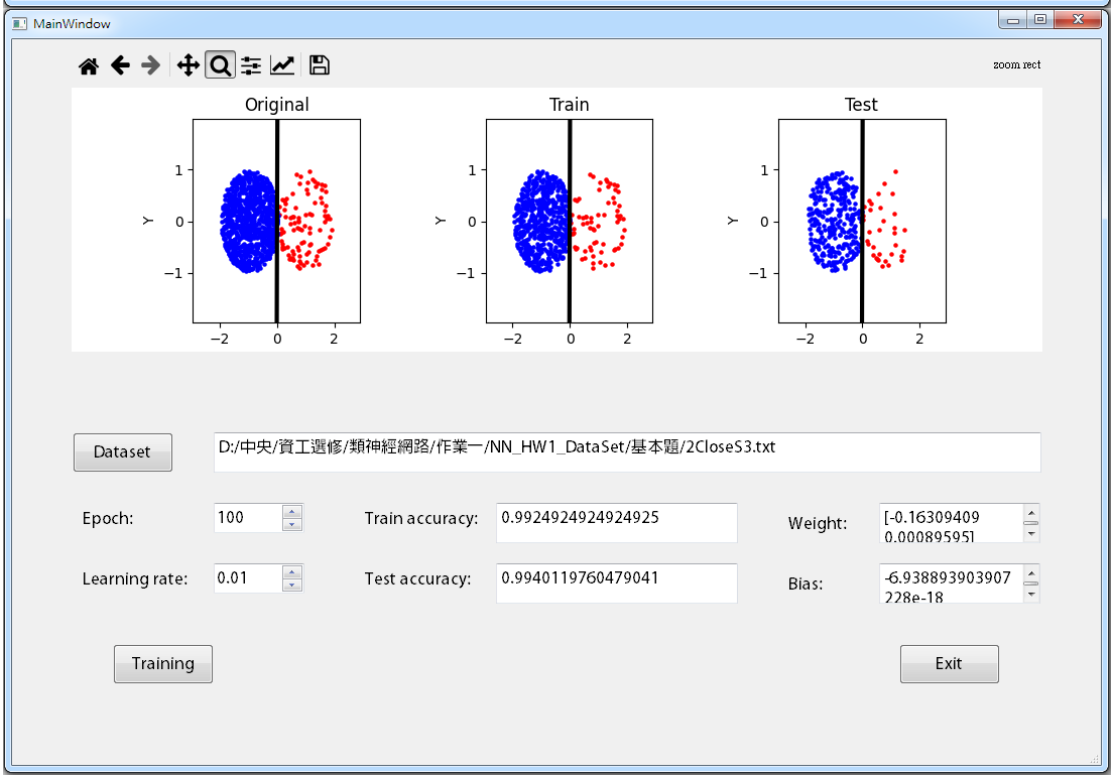

MainWindow
pan/zoom
Original
Train
Test
Dataset
D:/中央/資工選修/類神經網路/作業一/NN_HW1_DataSet/基本題/2CloseS2.txt
Epoch:
100
Train accuracy:
0.9849624060150376
Weight:
[-0.18243447
0.005926451
Learning rate:
0.01
Test accuracy:
0.9850746268656716
Bias:
-0.020000000000
000007
Training
Exit

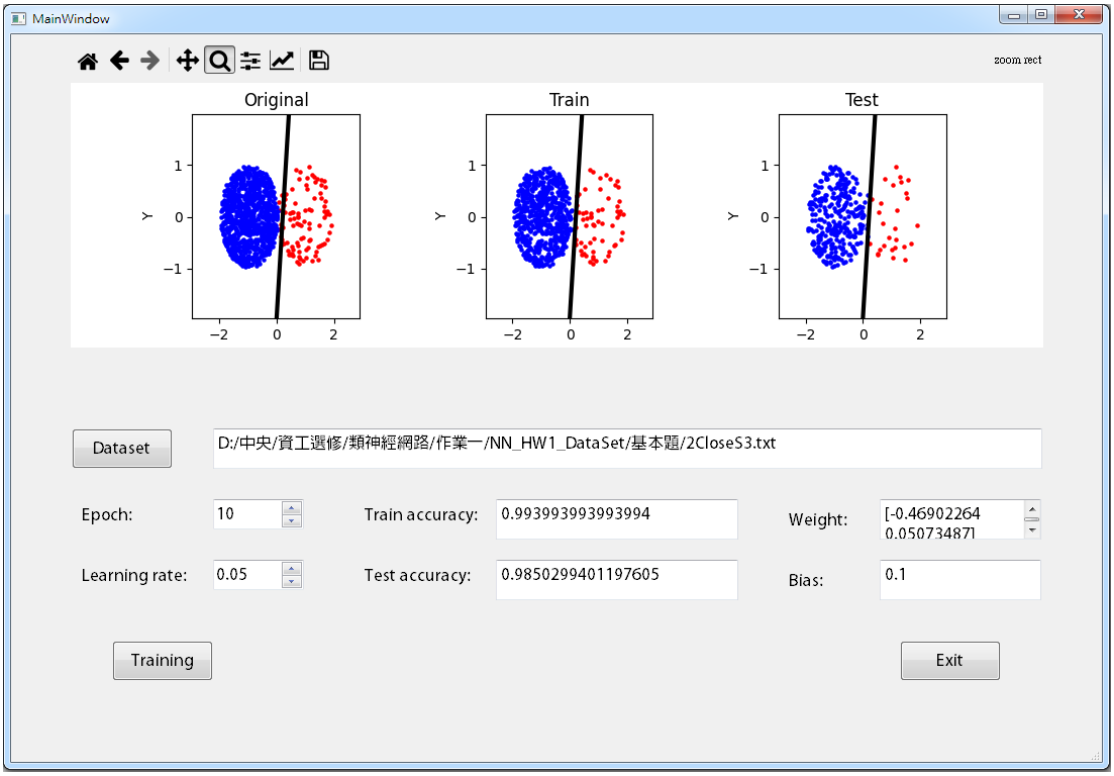


2CloseS2 也是線性可分割的，但由於兩群資料點靠得很近，所以有時會判斷錯誤，導致準確率不為 1。而也因為兩群資料很近的關係，epoch 次數小，就會對準確率影響很大，無法像 2CloseS 一樣輕鬆解決問題

7. 2CloseS3

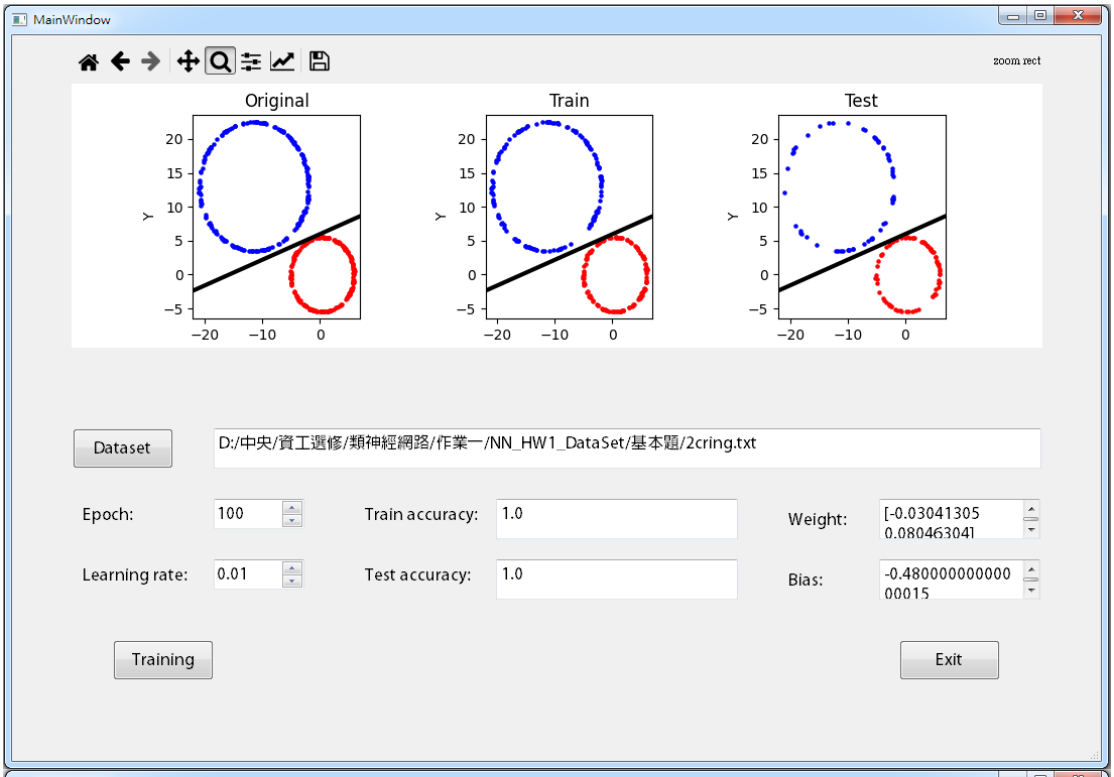

MainWindow
pan/zoom
Original
Train
Test
Dataset
D:/中央/資工選修/類神經網路/作業一/NN_HW1_DataSet/基本題/2CloseS3.txt
Epoch:
100
Train accuracy:
0.9984984984984985
Weight:
[-0.60092787
0.039943821
Learning rate:
0.01
Test accuracy:
1.0
Bias:
0.
01999999999999
Training
Exit

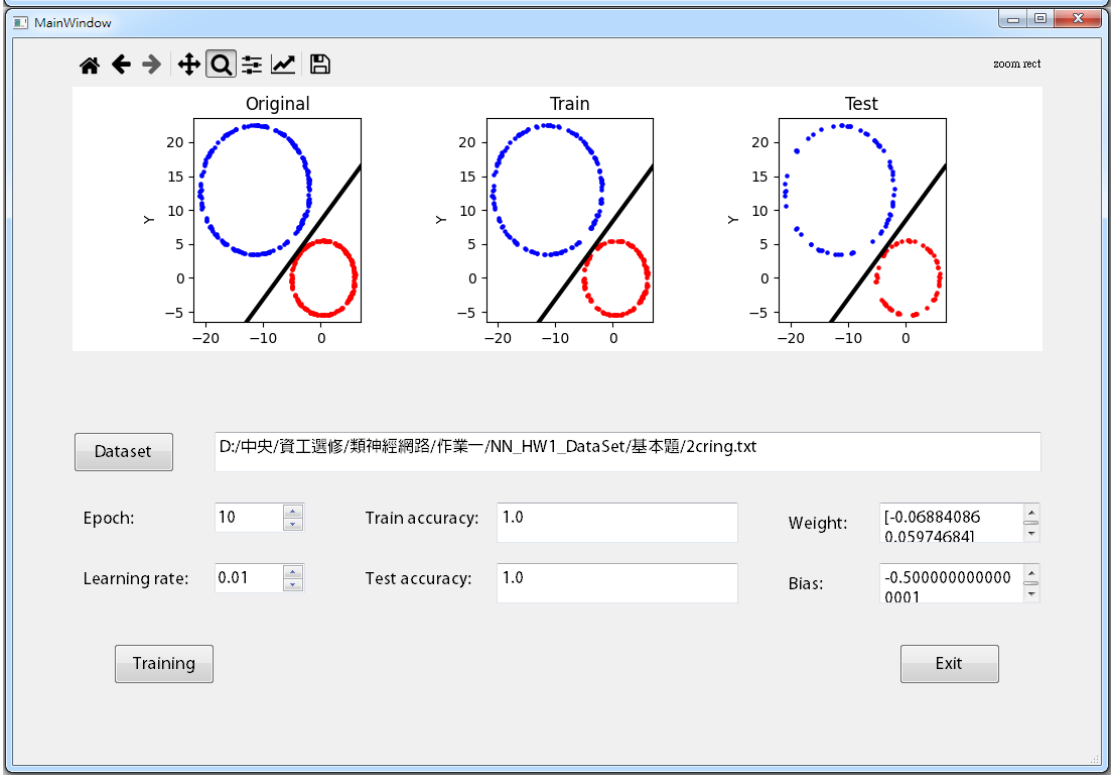

MainWindow
zoom rect
Original
Train
Test
Dataset
D:/中央/資工選修/類神經網路/作業一/NN_HW1_DataSet/基本題/2CloseS3.txt
Epoch:
100
Train accuracy:
0.9924924924924925
Weight:
[-0.16309409
0.000895951
Learning rate:
0.01
Test accuracy:
0.9940119760479041
Bias:
-6.938893903907
228e-18
Training
Exit



2CloseS3 幾乎可以完美分成兩群，但原始分類中，藍色點群中會混入一個紅色點，所以只能接近完美。除了那一點紅色點之外的點都離的不算近，所以 epoch 低、學習率高造成的影響沒有到很大

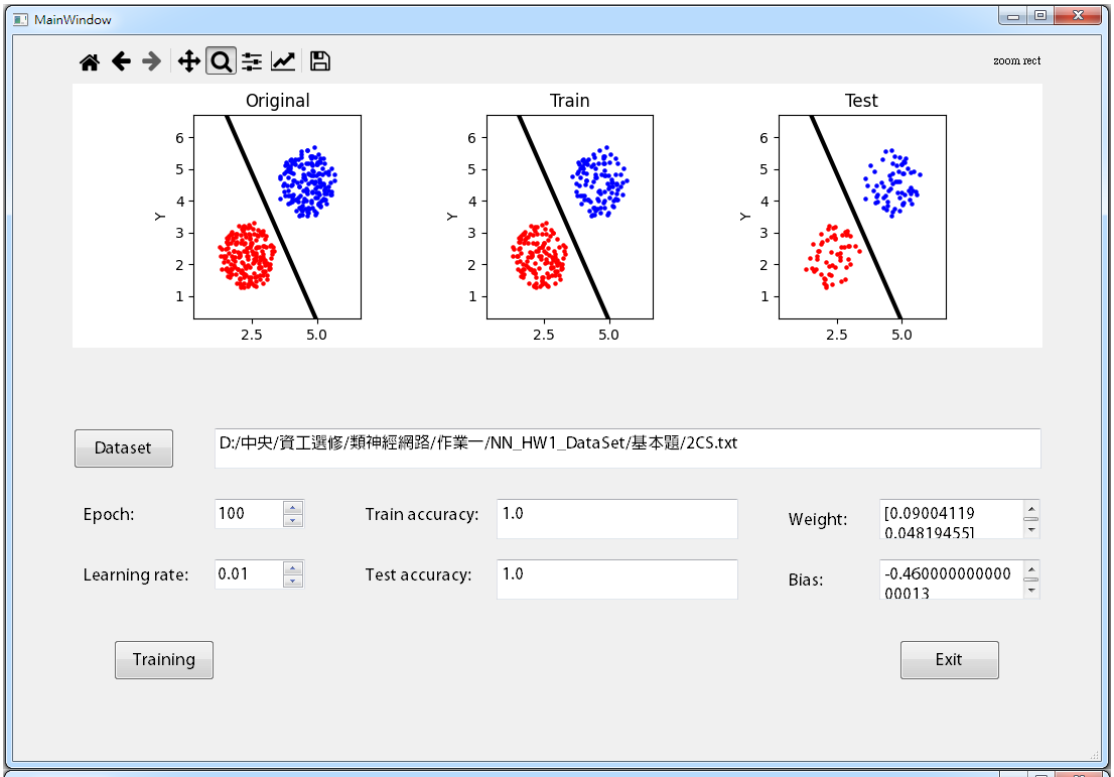
8. 2cring

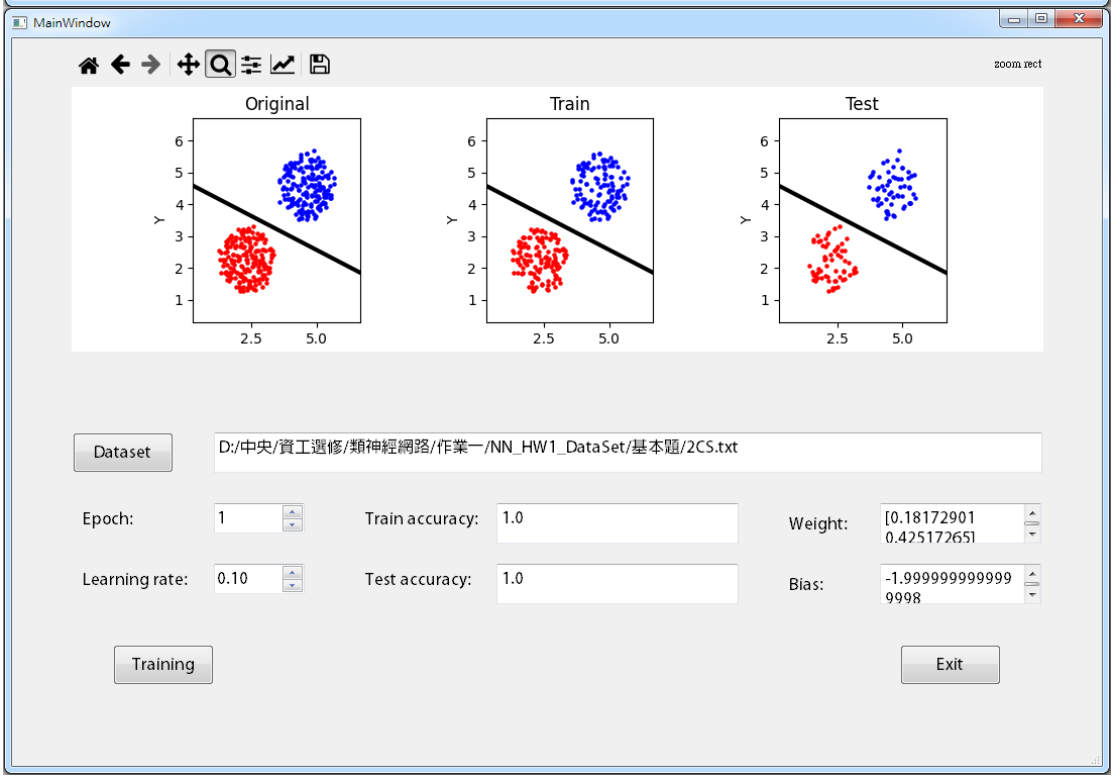




兩群點線性可分、形成的圓分的不近，且點的數量夠多，不太會有訓練資料缺少靠近另一群的部分導致分類錯誤的問題，所以 epoch 調低、學習率調高一些對準確率不會造成影響，通常準確率皆為 1

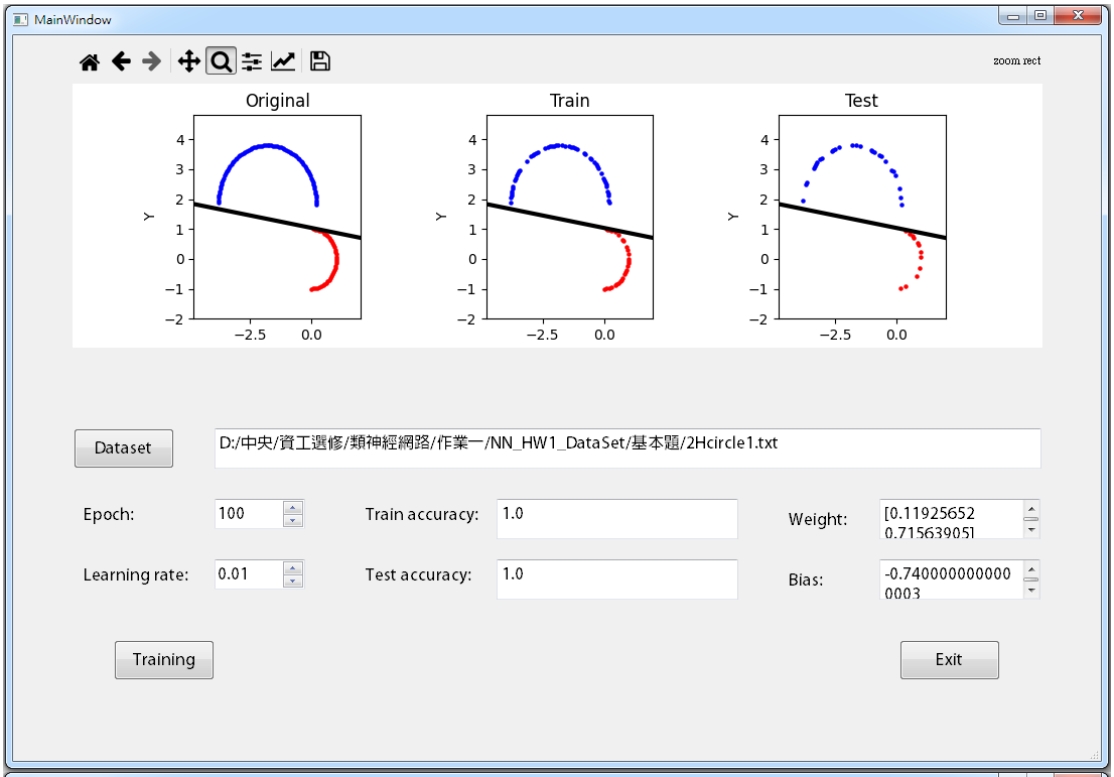
9. 2CS

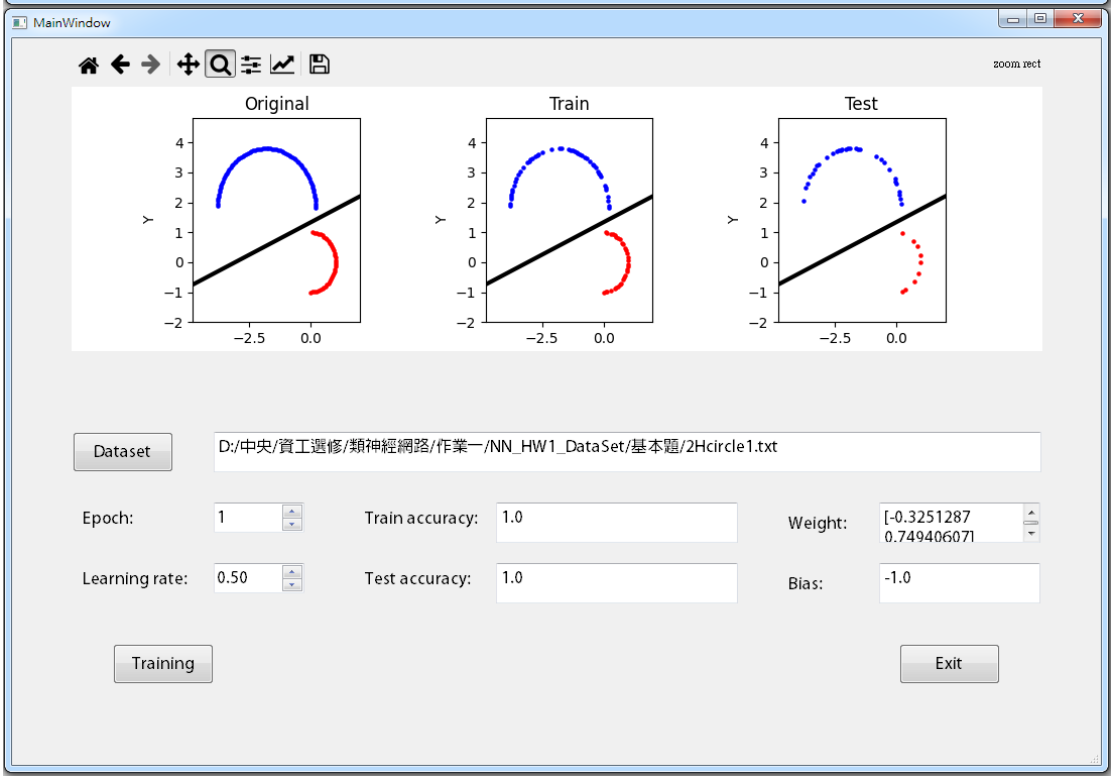


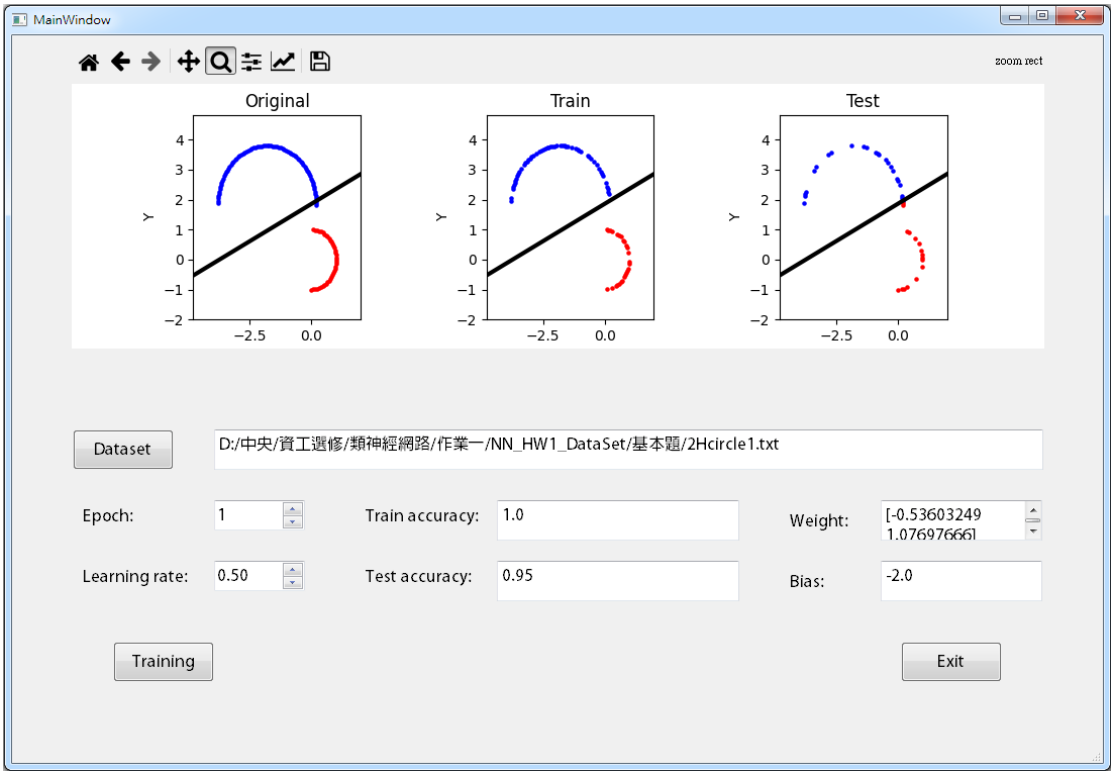


2CS 線性可分，且兩群點離得很遠，所以準確率基本上都是 1，就算 epoch 調到只剩 1 次，學習率調高到 0.1 也不受影響

10. 2Hcircle1

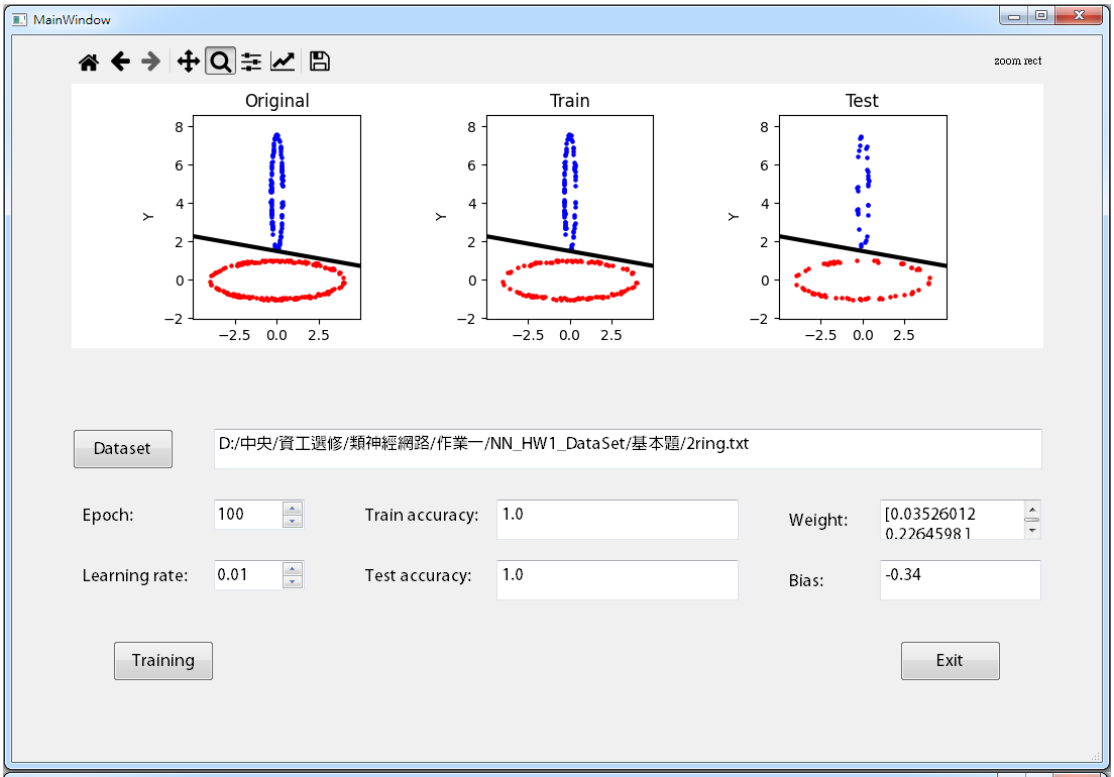


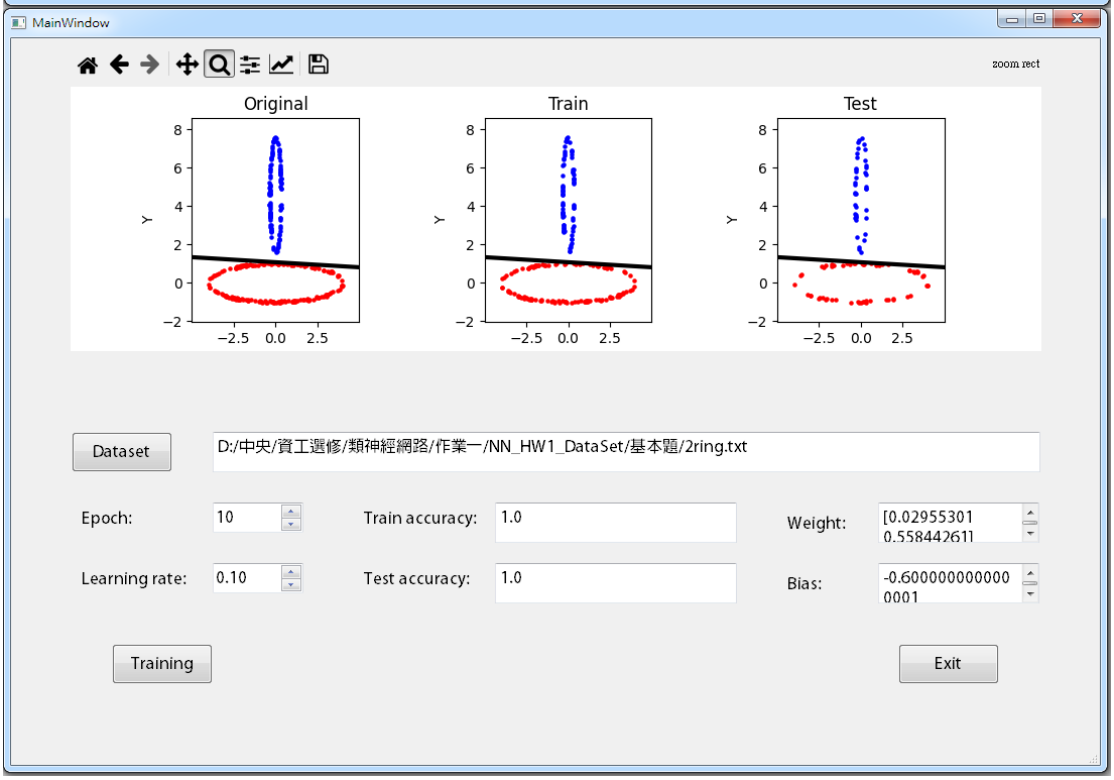




同 2CS，10. 2Hcircle1 線性可分，且兩群點離得不算近，準確率基本上都是 1，就算 epoch 調到只剩 1 次，學習率調高一些也不太會受影響，不過偶爾會因為訓練資料沒有靠近另一群的點，導致測試準確率不為 1

11. 2ring

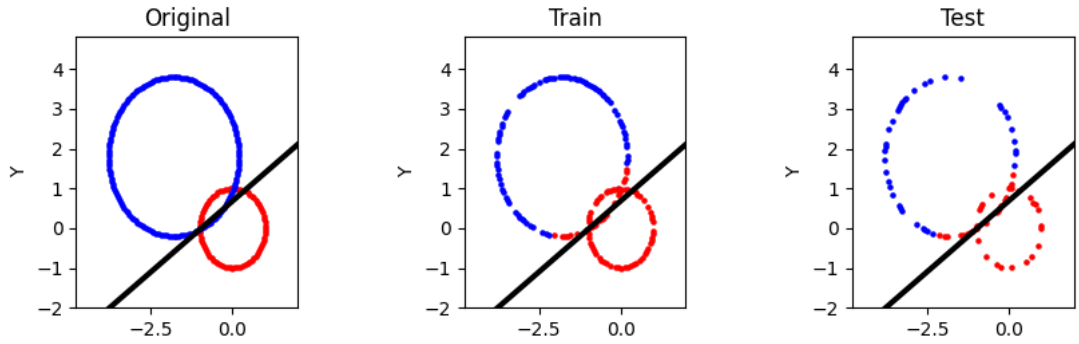




2ring 線性可分、兩群點離的不近，所以訓練及測試準確率基本上都是 1，就算 epoch 調到只剩 10 次，學習率調高到 0.1 也不太影響

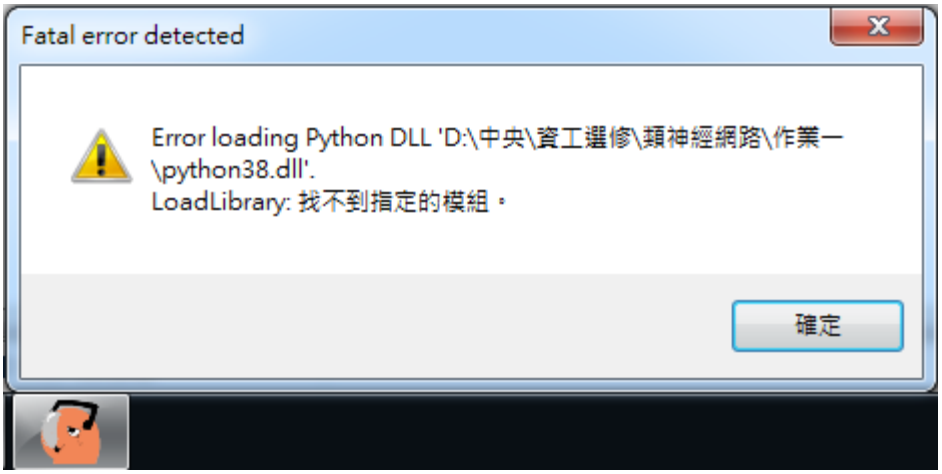
四、實作問題

1. 因為不熟悉 numpy 操作的關係，光宣告空的 nparray 就失敗了好幾次
2. 不熟悉 PyQt5 及物件導向的觀念，不論是找資料、修改程式、debug 都常常找不出問題，執行也常常只會出現 process finished with exit code 1073741845，在 debug 上花了不少時間，甚至有時候只是程式碼裡的物件命名和 UI 拉的不一樣，卻花了 3 個小時以上在除錯，基本上可以說實作遇到的問題和浪費的時間有七成都在這種地方上。
3. 感知機的預測線與分類出來的群沒有對到



不知為何，直接代算出來的 bias 下去算，畫出來的圖會非常詭異，不過將 bias 乘以兩倍畫出來就會正常。但是看了好幾次上課 PPT 及公式還是找不到問題所在。

4. Pyinstaller 產生的 exe 出現找不到指定的模組問題



原本打包 exe 的時候是參考網路上用 spec 檔案的教學，但產生出的檔案非常多，而且將 exe 移出原本的資料夾後就會發生找不到 dll 檔的問題。後來是用另一個方法才解決，雖然檔案比較大而且開啟時沒有更改的圖標了，但是比較實用一些，exe 檔傳到哪都可以直接執行。