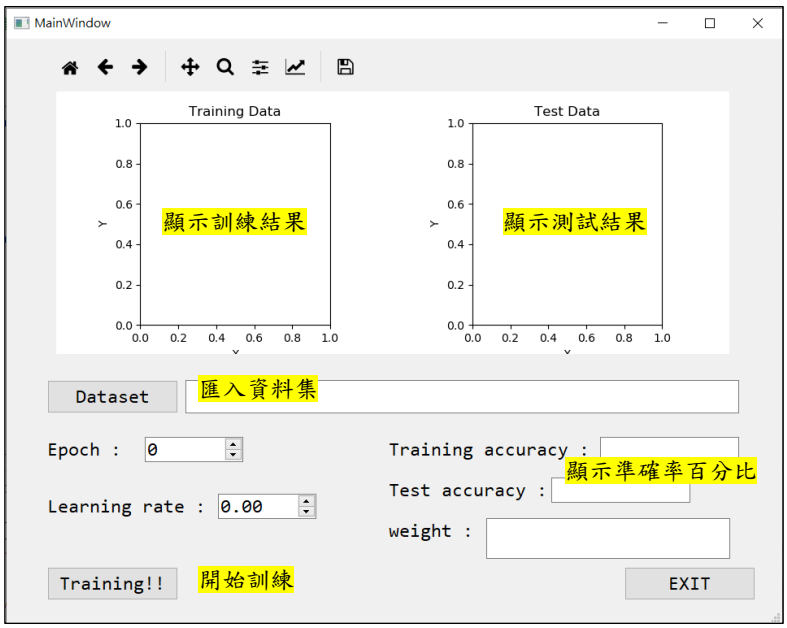
類神經作業一 — 設計感知機類神經網路書面報告

姓名：張芷烜　　系級：資工4A　　學號：109502510

A. 程式執行說明 (GUI功能說明)



➢ 使用Qt Designer 拉介面輸出.py檔

點擊Dataset匯入.txt 後，設定Epoch與Learning rate，點擊Training開始訓練，由上圖顯示訓練結果與測資資料分類結果，並且於下列顯示準確率與權重。

B. 程式碼簡介

- UI.py 圖形介面設定
- Main.py 主程式控制介面設定

➢ MainWindow -- _init_() 初始化設定

```
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        # 創建 UI
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.pts = np.empty([0, 2], float)
        self.pts_group = np.array([], int)
        self.groupID = np.array([], int)
        self.pred = np.array([], int)
        self.train_accuracy = 0

        # 連接按鈕的點擊事件到自訂的方法
        self.ui.dataset_button.clicked.connect(self.load_dataset)
        self.ui.training_button.clicked.connect(self.train_model)
        self.ui.exit_button.clicked.connect(self.exit_application)
```

初始設定button點擊對應控制函式

➢ load_dataset() 匯入資料集

```
def load_dataset(self):
    # 打開資料夾匯入資料集
    options = QFileDialog.Options()
    options |= QFileDialog.ReadOnly
    self.file_path, _ = QFileDialog.getOpenFileName(self, "Open .txt File", "", "Text Files (*.txt);;All Files (*)", options=options)
    self.ui.dataset_path.setText(self.file_path)
    font = QtGui.QFont()
    font.setFamily("Consolas")
    font.setPointSize(14)
    self.ui.dataset_path.setFont(font)
```

打開檔案管理

設定路徑顯示於UI

➢ train_model() 訓練資料

```
def train_model(self):
    self.reset()
    self.epoch = self.ui.epoch_spinBox.value()
    self.learning_rate = self.ui.learing_rate_spinBox.value()

    # print("epoch: ", self.epoch)
    # print("learning rate: ", self.learning_rate)

    # 讀取資料集
    with open(self.file_path, 'r') as file:
        datas = file.readlines()
        for data in datas:
            coord_x, coord_y, d = data.split(" ")
            # 把coord_x, coord_y 存成np.array
            self.pts = np.append(self.pts, np.array([[coord_x, coord_y]], dtype="float64"), axis=0)
            self.pts_group = np.append(self.pts_group, int(d))
            if int(d) not in self.groupID:
                self.groupID = np.append(self.groupID, int(d))
    np.sort(self.groupID)

    # 隨機將2/3的資料分成訓練集，1/3的資料分成測試集
    idx = np.random.permutation(len(self.pts)) # 打散順序
    self.train_size = int(len(self.pts) * (2 / 3))
    self.train_idx = idx[:self.train_size]  # 前2/3的資料當訓練集
    self.test_idx = idx[self.train_size:]  # 後1/3的資料當測試集

    # 預測訓練結果
    w = window.train_predict()
    self.test_predict(w)

    # 畫圖
    self.train_pic(w)
    self.test_pic(w)
    self.ui.widget.canvas.draw()
    # print("train accuracy: ", self.train_accuracy)
    formatted_accuracy = "{:.2f}%".format(self.train_accuracy * 100)
    self.ui.training_ac_label.setText(formatted_accuracy)
    font = QtGui.QFont()
    font.setFamily("Consolas")
    font.setPointSize(14)
    self.ui.training_ac_label.setFont(font)
    formatted_accuracy = "{:.2f}%".format(self.test_accuracy * 100)
    self.ui.test_ac_label.setText(formatted_accuracy)
    font = QtGui.QFont()
    font.setFamily("Consolas")
    font.setPointSize(14)
    self.ui.test_ac_label.setFont(font)
    self.ui.weight_label.setText(str(w))
    font = QtGui.QFont()
    font.setFamily("Consolas")
    font.setPointSize(10)
    self.ui.weight_label.setFont(font)
    # print("test accuracy: ", self.test_accuracy)
    # print("w: ", w)
```

切割資料對應 coord_x、coord_y、d
分別存入 self.pts / self.pts_group

分成訓練資料集與測試資料集

改變鍵結值函式與鍵結值預測函式

按照分類結果畫出圖

UI顯示準確率設定

➢ train_predict() 匯入資料集

```
def Activate_function(self,x): # 活化函數
    if x < 0:
        return self.groupID[0]
    else:
        return self.groupID[1]

def train_predict(self):
    w = np.random.uniform(0, 1, size=len(self.pts[0])) # 初始化權重 -1~1
    w = np.concatenate(([0], w), axis=0)
    print("w: ", w)


    for i in range(self.epoch): # 要迭代幾次
        for j in self.train_idx:
            input = self.pts[j]
            input = np.concatenate(([-1], input), axis=0)
            label = self.pts_group[j]
            output = np.dot(input,w)
            pred_result = self.Activate_function(output)
            # print(label, pred_result)
            w = w + self.learning_rate * (label - pred_result) * np.array(input)

    return w
```

活化函數

初始化權重(按照輸入維度+1)，初始化$W_{j_0} = 0$ (初始閾值θ)

輸入$X_{j0} = -1$

權重與輸入內積

$$\underline{w}(n+1) = \begin{cases} \underline{w}(n) + \eta \underline{x}(n) & \text{如果} \underline{x}(n) \in C_1 \text{和} \underline{w}^T(n)\underline{x}(n) < 0 \\ \underline{w}(n) - \eta \underline{x}(n) & \text{如果} \underline{x}(n) \in C_2 \text{和} \underline{w}^T(n)\underline{x}(n) \geq 0 \\ \underline{w}(n) & \text{如果} \underline{x}(n) \text{被正確分類} \end{cases}$$

根據活化函數結果更新權重

➢ test_predict() 測試資料分類

```
def test_predict(self, w):
    for i in self.train_idx:
        input = self.pts[i]
        input = np.concatenate(([-1], input), axis=0)
        output = np.dot(input,w)
        pred_result = self.Activate_function(output)
        self.pred = np.append(self.pred, pred_result)

    # print("train pred: ", self.pred)

    for i in self.test_idx:
        input = self.pts[i]
        input = np.concatenate(([-1], input), axis=0)
        output = np.dot(input,w)
        pred_result = self.Activate_function(output)
        self.pred = np.append(self.pred, pred_result)

    # print("test pred: ", self.pred[len(self.train_idx):])

    return
```

使用最後的權重對train_data和test_data進行預測分類

➢ train_pic() / test_pic() 繪製結果圖

```
def train_pic(self, w):
    self.ui.widget.canvas.train.cla() # 清除畫布
    for i in range(len(self.train_idx)):
        j = self.train_idx[i]
        if self.pred[i] == self.groupID[0]:
            self.ui.widget.canvas.train.scatter(self.pts[j, 0], self.pts[j, 1], s=5, color='r')
        else:
            self.ui.widget.canvas.train.scatter(self.pts[j, 0], self.pts[j, 1], s=5, color='b')

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

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

    slope = -w[1] / w[2]
    intercept = w[0] / w[2]
    print("slope: ", slope, "intercept: ", intercept)
    x = np.array([min(self.pts[:, 0]) - 1, max(self.pts[:, 0]) + 1])
    y = slope * x + intercept
    print("x",x,"y",y)
    # 绘制直线
    self.ui.widget.canvas.train.plot(x, y, color='k', lw=3)

    # 设置坐标轴范围
    self.ui.widget.canvas.train.axis(xmin=min(self.pts[:, 0]) - 1, xmax=max(self.pts[:, 0]) + 1)  # 設定x軸顯示範圍
    self.ui.widget.canvas.train.axis(ymin=min(self.pts[:, 1]) - 1, ymax=max(self.pts[:, 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")
```

分兩類標記不同顏色

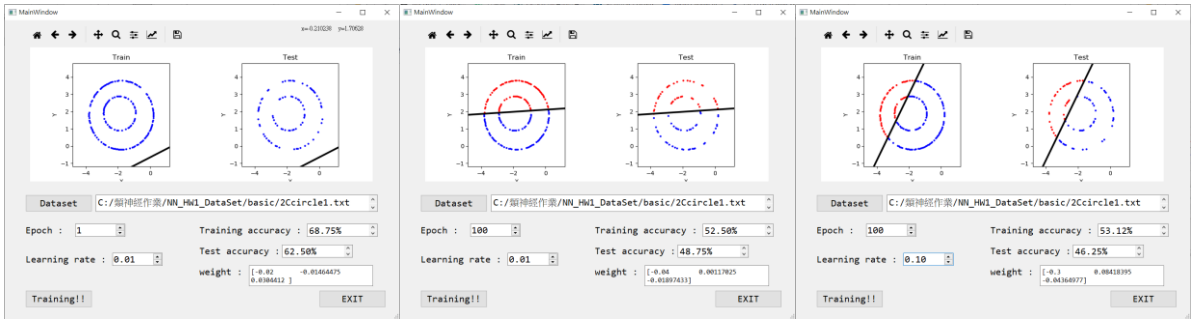
如果預測節果與期望結果相同 AC+1

計算準確率百分比

計算斜率與截距繪製分類線

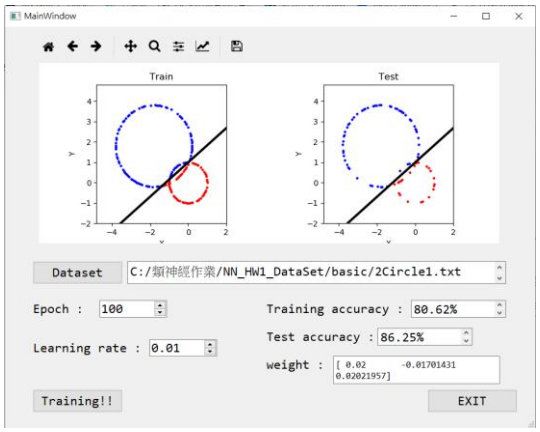
C. 實驗結果

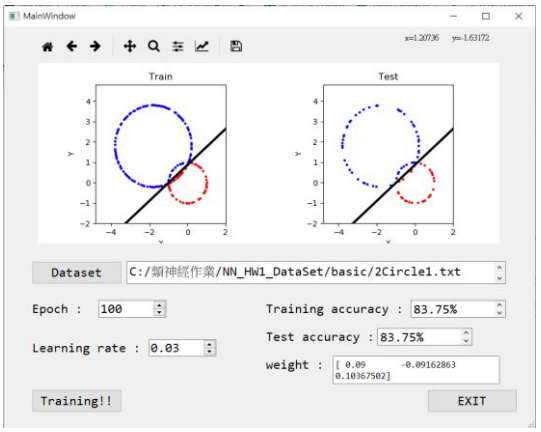
1. 2Ccircle1.txt



感知機對於非線性分類無法收斂，因此不管迭帶幾次，學習率怎麼設定，準確率都很差，基本50%就是用猜的。

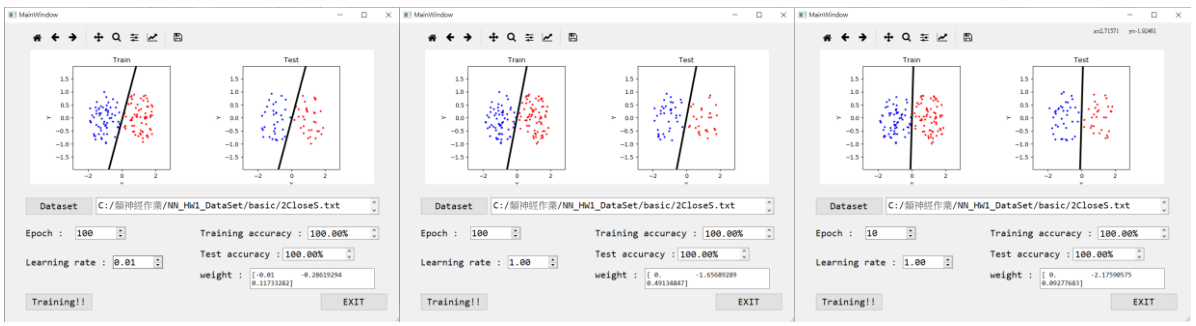
2. 2Circle1.txt





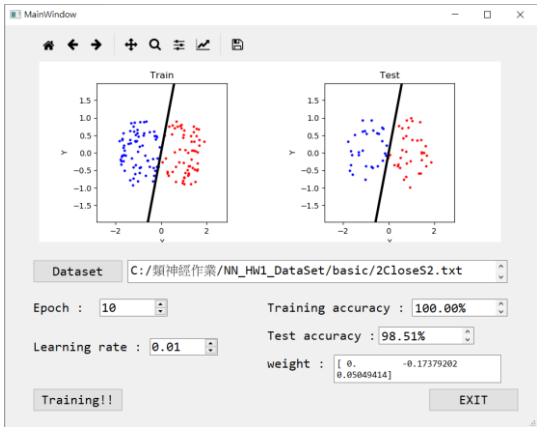
準確率相較 2Ccircle1.txt 高了許多，雖然同為非線性分類，但可以看出除了交疊處無法線性分割，大致上還是可以藉由線分兩邊。但就算再調整參數，準確率也無法提升了。

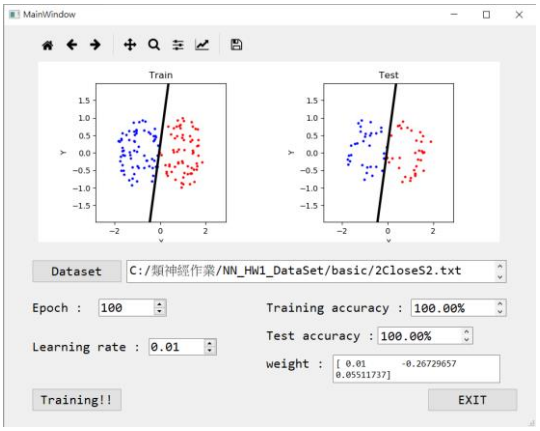
3. 2CloseS.txt



2CloseS 是線性可分分佈，因此不管學習率設大一點或是跌代次數很少，都有很高的準確率。

4. 2CloseS2.txt



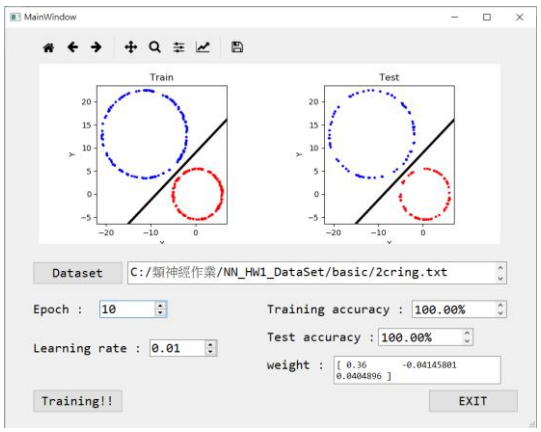


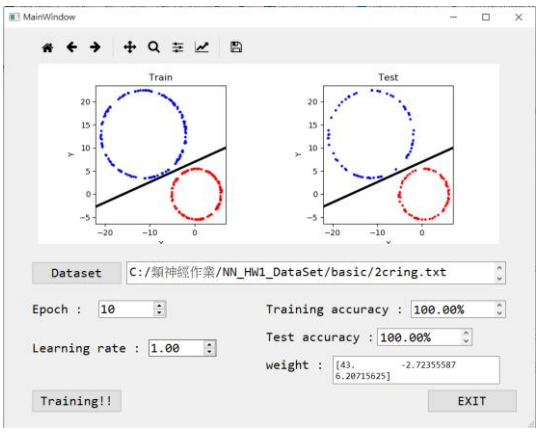
與 2CloseS 相同 都有很高的準確率，但資料集距離太近，有時無法百分之百正確切分，需要迭代次數高找到最正確的權重。

5. 2CloseS3.txt

迭代次數不用很高就有很好的準確率了，可以觀察出資料集分佈的很明確清晰，但不管epoch調到多少，準確率都沒辦法兩著都達到100%，應該是一開始的期望輸出有label錯誤。

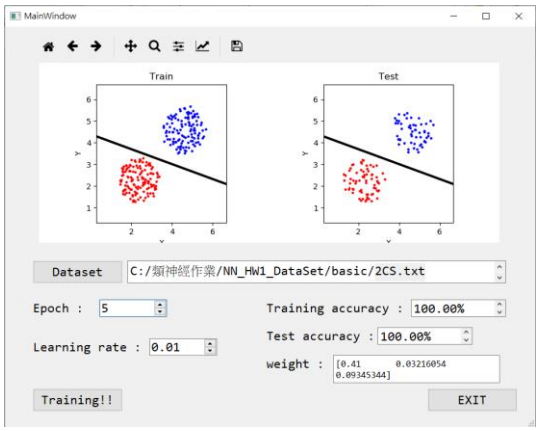
6. 2cring.txt

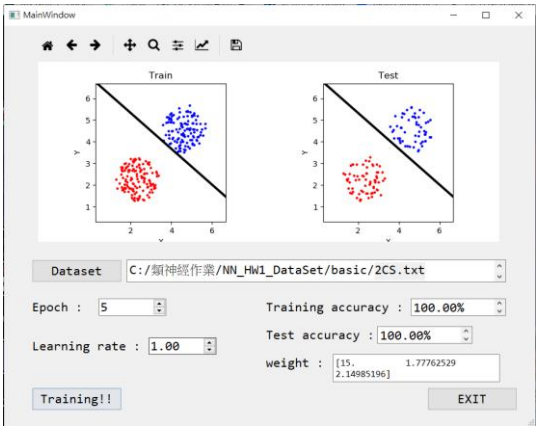




線性可分且兩者分佈距離，資料量足夠，可以很正確的切分，即使學習率很高，只要有足夠的epoch次數就可以調整到正確的鍵結值。

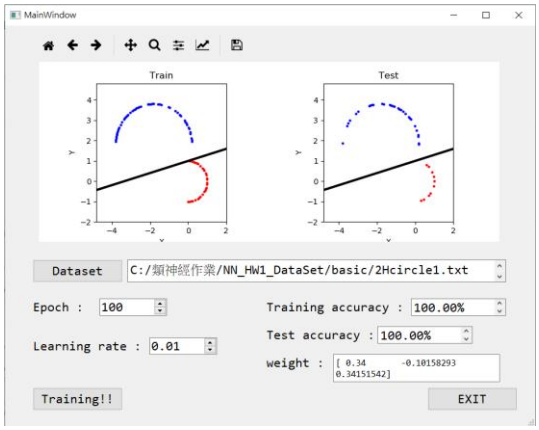
7. 2CS.txt

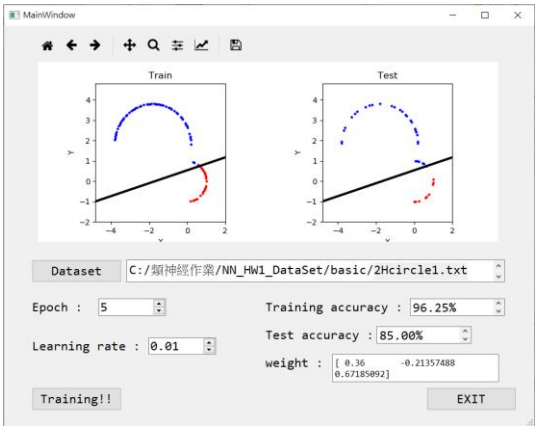




線性可分距離又夠，就算迭代很少學習率很大準確率也能百分之百。

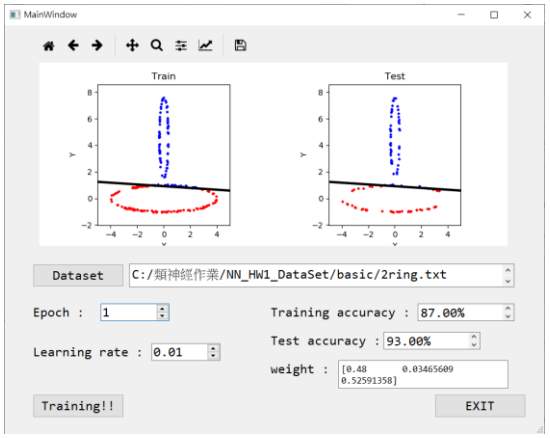
8. 2Hcircle1.txt

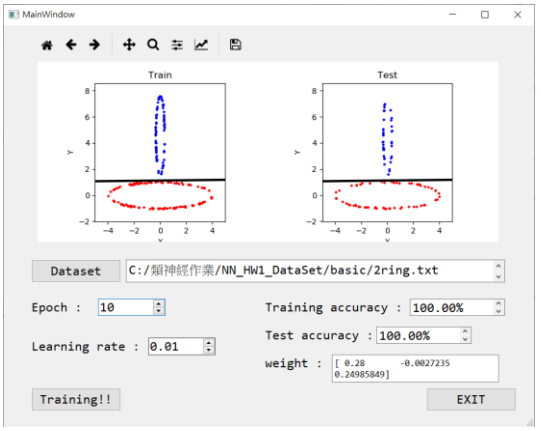




也可以很好的線性分割，但有時候資料有斷層會有小錯誤。

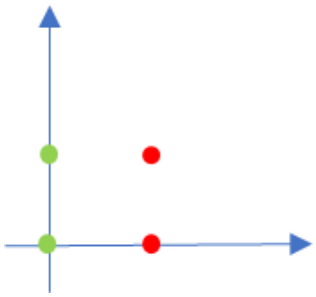
9. 2ring.txt

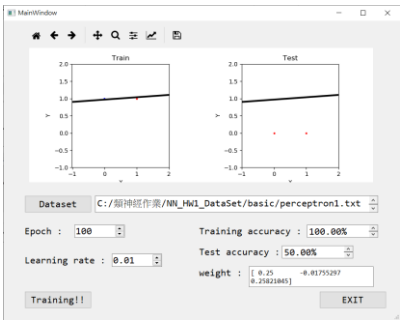


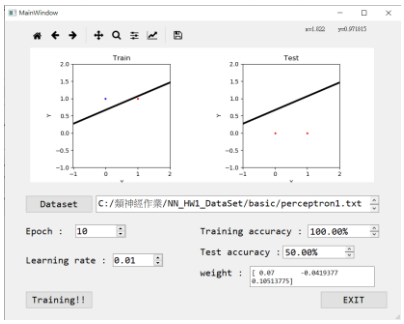


線性可分，但找到的線有時候太接近下面圓的切線，再交界處的有時候會被分類錯誤，除此之外都有很高的準確率。

9. perceptron1.txt

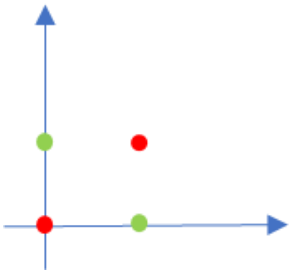


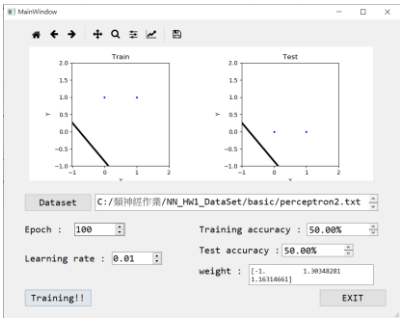


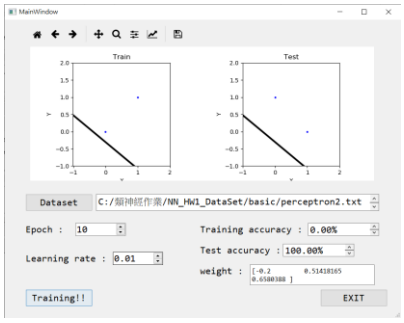


總共資料集只有四點，就算epoch調大，學習率調很小孩是找不到正確分割線。

10. perceptron2.txt







總共資料集只有四點，且分類無法找到直線分割(類似XOR問題)，無法正確分類。

D. 實驗結果分析及討論

若資料是線性可分的資料集，不管學習率設多少，都可以收斂到最後正確分類，不過為了怕錯過正確線將學習率調低相對的epoch次數就要多，但資料如果明確可分割成兩類時，就算epoch次數很少還是能達到很好的準確率。反觀因為單層感知機計算權重一分為二的關係，再非線性可分的資料集中，線段會不段跳躍，好像在猜測一般測試最好的答案。

在上面基礎題的測試可以發現，若是資料本身沒有很好的分割性，找不出統一規則時，例如上面的 2Circle1.txt，雖然大致上可以分類但是即便訓練次數很高，學習率很小，還是找不到正確解，所以單層感知機的分類能力有限，若需要完成更高準確率的分類還是要透過多層+倒傳遞演算法。