類神經網路 HW1 感知機實驗報告

資工碩一 113522053 蔡尚融

目錄

一.　程式執行說明

報告內容

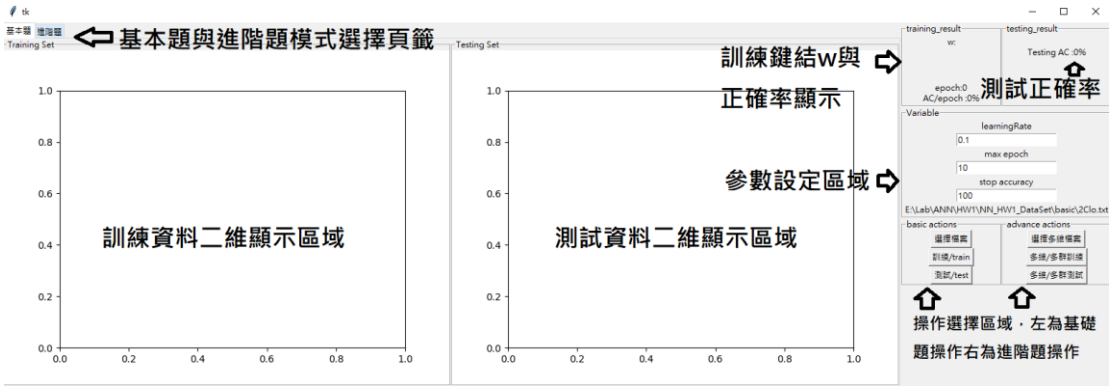
此程式作業實作包含:

基礎題

加分題

a. 三維資料圖形顯示介面。
b. 能夠處理多維資料(四維以上)。
d. 可以辨識兩群以上的資料。

基礎題操作介面:



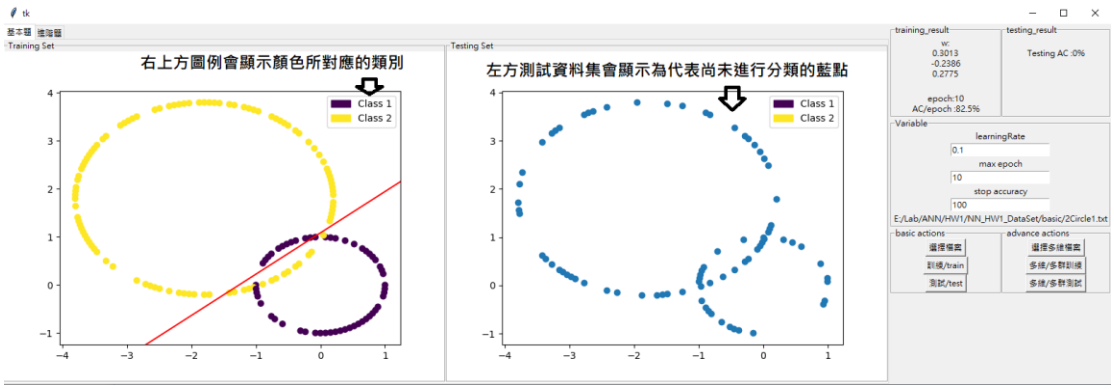
操作步驟:

Step1: 於左上方選擇基礎題頁籤。

Step2: 於右方參數設定區域分別設定 learningRate(學習率)、max epoch(epoch 達到多少停止訓練)、stop accuracy(達到多少準確率停止訓練)。

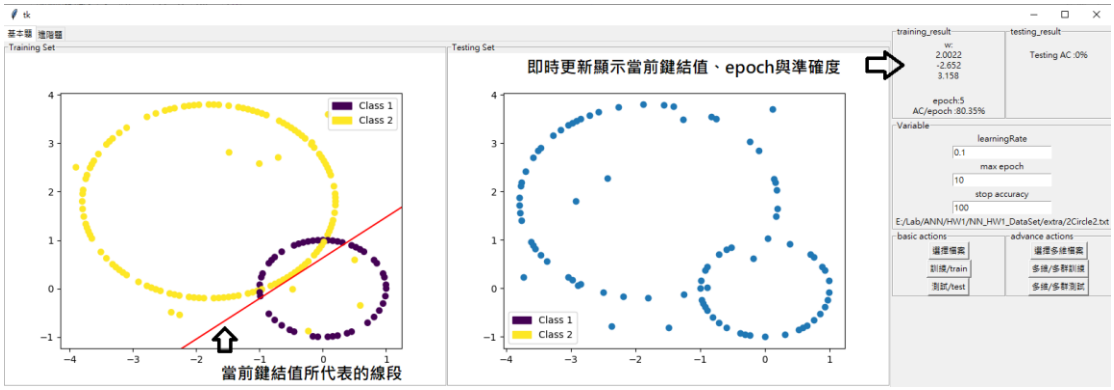
Step3: 點擊右下方 basic actions 中的”選擇檔案” Button。

此時畫面會顯示選取檔案之資料點如下:



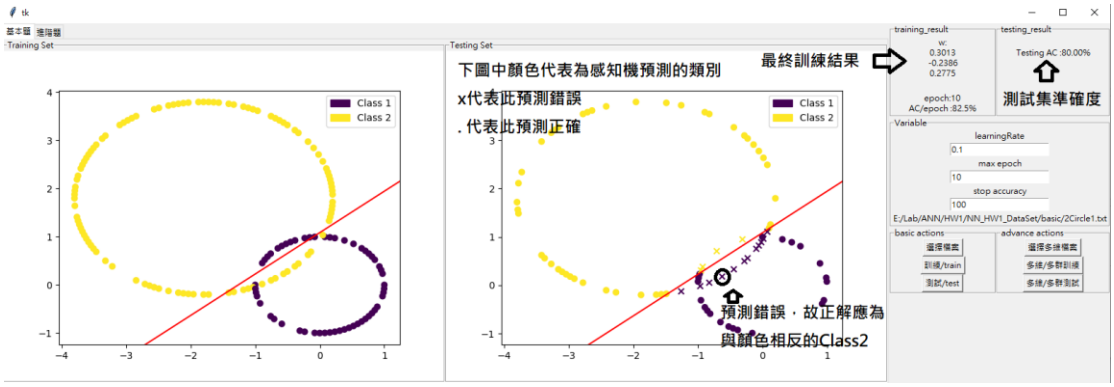
Step4: 點擊右下方 basic actions 中的 ”訓練/train” Button。

此時畫面會即時顯示訓練中狀態如下(示意圖不小心選到 extra 中的題目，故有雜點，但不影響說明)



Step5: 待訓練完成，點擊右下方 basic actions 中的 ”測試/Test” Button 進行測試。

結果畫面將顯示預測判別結果如下:

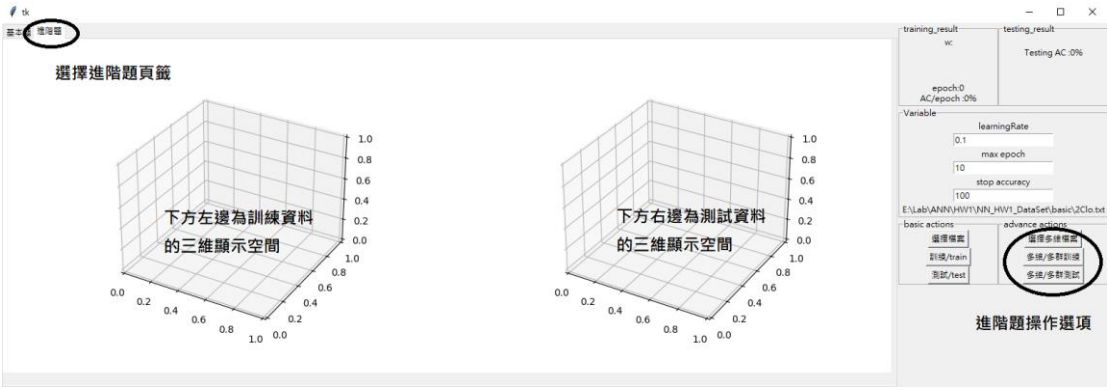


Testing Set 圖形顯示區域中顏色代表感知機的預測結果，符號”x”代表此筆

預測為錯誤的(正確類別與顏色相反)，符號 ” . ” 代表此筆預測為正確的(正確類別與顏色相同)。

進階題操作介面:

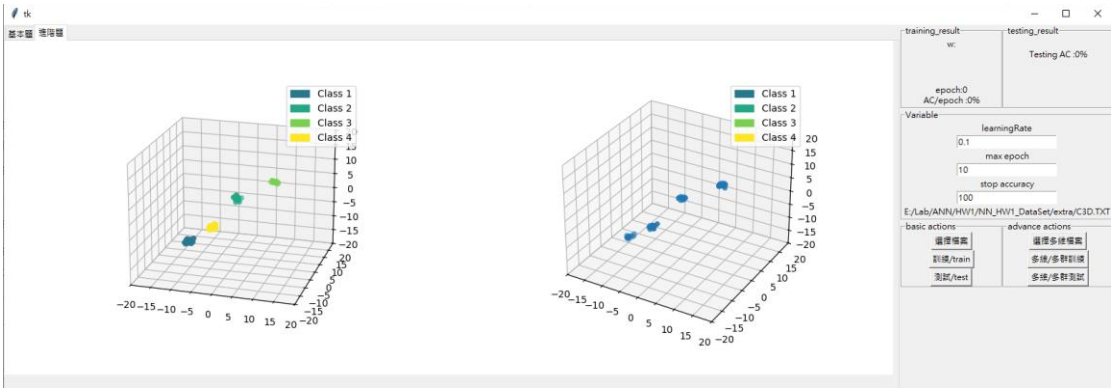
Step1:於上方頁籤中選擇進階題頁籤。



Step2: 於右方參數設定區域分別設定 learningRate(學習率)、max epoch(epoch 達到多少停止訓練)、stop accuracy(達到多少準確率停止訓練)。

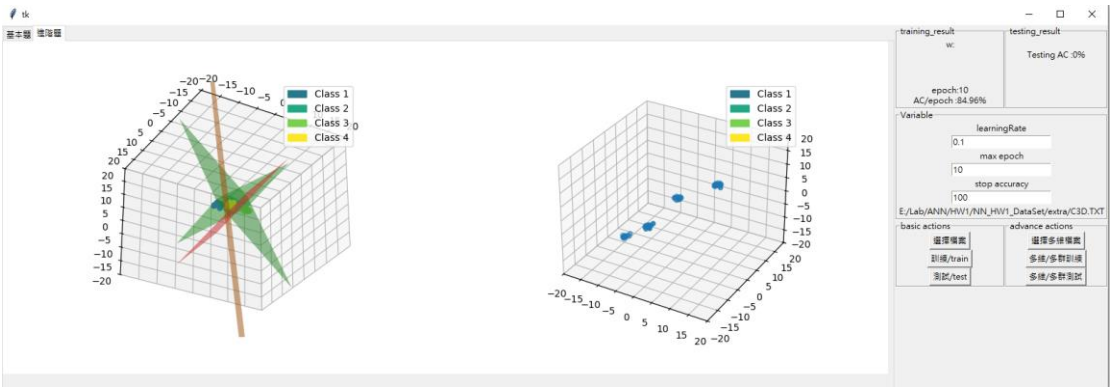
Step3: 點擊右下方 Advance actions 中的”選擇多維檔案” Button。

左方顯示區域會展示 3 維以下(3 維或 2 維的資料圖形)。

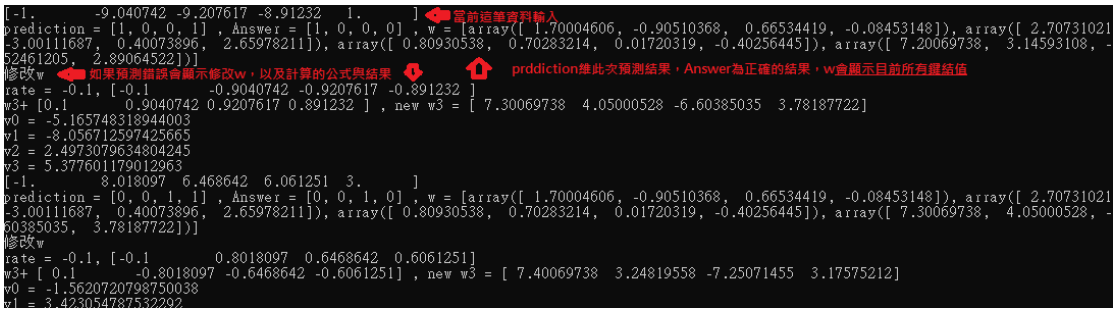


Step4: 點擊右下方 Advance actions 中的 ” 多維/多群訓練” Button。

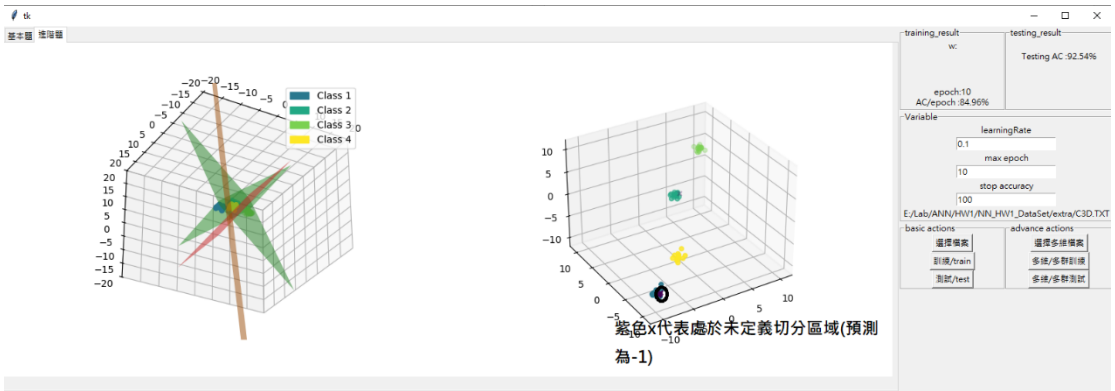
會顯示使用多個面/線段對資料進行切割的結果，並於右上方顯示正確率。



(若超過 4 維資料不會顯示圖形，會於右上方 Traning Result 中顯示測試準確率，並於 Consol 視窗中顯示鍵結值)



Step5: 待訓練完成，點擊右下方 basic actions 中的 ” 多維/多群測試” Button 進行測試。



(若超過 4 維資料不會顯示圖形，會於介面右上方 Result Area 中顯示測試準確率，並於 consol 中顯示 Code:每個感知機預測的結果、prediction:該 code 結果傳換為預測類別後的數值，以及該筆資料正確的類別)。

二.　程式碼簡介

基本題:

Select():

介面中 Basic action 下的選擇檔案 Button 會觸發 select() function，此 function 的主要工作為透過 np.loadtxt()去讀取所選的檔案，並讀取 data[: , 2]中的所有類別資料來判斷總共有哪些類別 cata(用在確認類別總數、以及進階題決定需要幾個感知機)。

這個 Function 會接著呼叫 split_data()、data_update()

```
#選擇檔案按鈕
def select():
    global data,train_data,test_data,data_len,cata
    TrainingAC.set("epoch:0\nAC/epoch :0%")
    TestingAC.set("\nTesting AC :0%")
    FP.set(filedialog.askopenfilename())
    data = np.loadtxt(FP.get())
    data_len = len(data)
    cata = data[:,2]
    cata = np.unique(cata)
    cata = np.sort(cata)
    print(cata)

    split_data()
    data_update()
```

Split_data():

透過 random 來隨機選取資料中的 2/3 的 index 作為 training data，剩下的 1/3 資料作為 test data。

```
#切分訓練/測試集
def split_data():
    global train_data,test_data
    np.random.seed()
    train_data_index= random.sample(range(data_len),int(data_len/3*2))
    #print(train_data_index)
    train_data = data[train_data_index,:]
    test_data = np.delete(data,train_data_index,axis=0)
```

Data_update():

此 function 的主要功能是在選擇新的資料時或重新開始訓練時將圖表進行初始化，包括將資料點繪製在 Traning Set 以及 Testing Set 兩張二維圖表上，並未兩張圖表都加上圖例 legend。

```
#更新圖表dots
def data_update():
    global line,line2,dots
    x=train_data[:,0]
    y=train_data[:,1]
    c = train_data[:,2]
    #print(test_data)
    ax.cla()
    dots=ax.scatter(x,y,c=c,cmap='viridis', vmin=cata[0], vmax=cata[1])
    line, = ax.plot([0,0], [0,0], color='red', label='線')

    # 創建對應 class 的 Patch，用來顯示 legend
    class_0_patch = Patch(color=dots.cmap(dots.norm(int(cata[0]))), label=f'Class {int(cata[0])}')
    class_1_patch = Patch(color=dots.cmap(dots.norm(int(cata[1]))), label=f'Class {int(cata[1])}')
    # 添加 legend，並將 class 0 和 class 1 對應顏色添加到圖例中
    ax.legend(handles=[class_0_patch, class_1_patch])


    canvas.draw()
    canvas.flush_events()
    time.sleep(0.1)

    x=test_data[:,0]
    y=test_data[:,1]
    c = test_data[:,2]
    #print(test_data)
    ax2.cla()
    ax2.scatter(x,y,cmap='viridis')
    line2, = ax2.plot([0,0], [0,0], color='red', label='線')
    ax2.legend(handles=[class_0_patch, class_1_patch])
    canvas2.draw()
    canvas2.flush_events()
    time.sleep(0.1)
```

W_update():

此 function 的主要目的是讓程式能夠在感知機訓練時能夠將鍵結值所代表的線段動態更新至圖表上，主要透過對於 data_update()中宣告的 line2D 物件 line 進行 line.set_data()操作來進行線段更新。

```
#更新圖表預測線段
def w_update(w,line,canvas):
    c,a,b = w
    W.set(f"w:\n{float(c):.4f}\n{a:.4}\n{b:.4}\n")

    c=c*-1
    # 設定 x 的範圍，例如從 -10 到 10
    x_vals = np.linspace(-10, 10, 2)
    # 計算 y 的對應值
    y_vals = - (a / b) * x_vals - (c / b)
    line.set_data(x_vals,y_vals)
    #plt.plot(x_vals, y_vals, label=f'{a}x + {b}y +
    canvas.draw()
    canvas.flush_events()
    time.sleep(0.1)
    return line
```

training():

Training function 中包含了主要的感知實作(主要步驟如下):

1. 首先透過 w = np.random.rand(3) * 0.01 將鍵結值以及 bias 進行隨機初始化
2. 呼叫 w_update 將初始化的鍵結直線段繪製至圖表上。

```
np.random.seed()
w = np.random.rand(3) * 0.01
#w = np.insert(w,0,-1)
print(f"randomw = {w}")
w_update(w,line,canvas)
```

3. 接著進入 epoch 迴圈，在 epoch 迴圈下用 correct 變數計算正確率(初始化為 train data 的資料筆數，每當錯誤時-1)，接下來會開始迭代每一筆 trainging data 的迴圈，在這個迴圈中，會於輸入資料前方新增-1 用於與 bias 相乘，接著計算 v = np.dot(w,t_data[0:3])也就是 w 與輸入資料的內

積，並用 y = cata[1] if v>0 else cata[0] if v<0 else 0 來計算出該筆資料的預測結果是哪個種類。接下來程式碼透過 if y !=t_data[3]:來檢查此筆預測是否正確，若預測結果錯誤則對 w 的值進行修改，先透過 rate = learningRate if y < t_data[3] else -learningRate 來判斷修正的方向，並 w = w + rate*t_data[0:3]讓 w 等於 learning rate 乘上該筆輸入的資料。

```
for i in range(epoch):

    print(f"epoch = {i+1}")
    correct = len(train_data)
    for t_data in train_data:
        t_data = np.insert(t_data,0,-1)
        v = np.dot(w,t_data[0:3])
        print(f"v = {v}")
        y = cata[1] if v>0 else cata[0] if v<0 else 0
        #print(v)
        print(t_data)
        print(f"prediction = {y} , Answer = {t_data[3]} , w = {w}")
        if y !=t_data[3]:
            correct -= 1
            print("修改w")
            rate = learningRate if y < t_data[3] else -learningRate
            w = w + rate*t_data[0:3]
            print(f"rate = {rate}, {learningRate*t_data[0:3]}")
            print(f"w+ {rate*t_data[0:3]} , new w = {w}")
            w_update(w,line,canvas)
    train_accuracy.append(correct/len(train_data)*100)
    TrainingAC.set(f"epoch:{i+1}\nAC/epoch :{train_accuracy[-1]:.4}%")
    print(f"epoch: {i+1} 訓練正確率Accuracy{train_accuracy[-1]}%")

    if train_accuracy[-1] >=Acuracy:
        break
print(train_accuracy)
```

Testing():

此 function 於使用者點擊介面中 basic aciton 下，”訓練/Train” Button 時觸發，會透過 test_result 矩陣紀錄所有 test_data 迭代進行 v = np.dot(w,t_data[0:3])

矩陣內積的結果(同樣會於每筆輸入前插入-1 用於與 bias 項相乘)。

```
test_result = []
for t_data in test_data:
        t_data = np.insert(t_data,0,-1)
        v = np.dot(w,t_data[0:3])
        print(f"v = {v}")
        y = cata[1] if v>0 else cata[0] if v<0 else 0
        test_result.append(y)
        #print(v)
        print(t_data)
        print(f"prediction = {y} , Answer = {t_data[3]} , w = {w}")
```

接著將 test_result 中的所有預測結果與正確答案進行比對得到 correct_index 矩陣，最後將 test_data[correct_index]用 scattor ”o” 的符號以及預計的顏色繪製至圖表上並讓不正確的點透過 dots3 = ax2.scatter(np.delete(x,correct_index,axis=0),np.delete(y,correct_index,axis=0),c=np.delete(test_result,correct_index,axis=0),marker="x",cmap='viridis',)以符號” x ” 繪製至圖表上。

```
correct_index = []
for v,k in enumerate(test_result):
    if test_data[v,2]==k:
        correct_index.append(v)
TestingAC.set(f"\nTesting AC :{len(correct_index)/len(test_data)*100:.2f}%")
test_result = np.array(test_result)
ax2.cla()
class_0_patch = Patch(color=dots.cmap(dots.norm(int(cata[0]))), label=f'Class {int(cata[0])}')
class_1_patch = Patch(color=dots.cmap(dots.norm(int(cata[1]))), label=f'Class {int(cata[1])}')
ax2.legend(handles=[class_0_patch, class_1_patch])

dots2 = ax2.scatter(x[correct_index],y[correct_index],c=test_result[correct_index],marker="o",cmap='viridis', vmin=cata[0], vmax=cata[1])
dots3 = ax2.scatter(np.delete(x,correct_index,axis=0),np.delete(y,correct_index,axis=0),c=np.delete(test_result,correct_index,axis=0),marker="x",cmap='viridis', v
line2,=ax2.plot([0,0], [0,0], color='red', label='線')

w_update(w,line2,canvas2)
```

進階題:

Select_multi():

介面中 advance action 下的 ”選擇多維檔案” Button 會觸發 select_multi() function，此 function 的實作與基礎題主要差在新增了透過 cata(種類總數)來設定 n(總共要使用幾個感知機)的判別式，並透過 data.shape 判斷如果維度小於 4 再呼叫 ndGraphInitialize()進行三維圖表的初始化。

```python
#選擇檔案按鈕
def select_multi():
    global data,train_data,test_data,data_len,cata,n,dim
    TrainingAC.set("epoch:0\nAC/epoch :0%")
    TestingAC.set("\nTesting AC :0%")
    FP.set(filedialog.askopenfilename())
    data = np.loadtxt(FP.get())
    data_len = len(data)
    cata = data[:,-1]
    cata = np.unique(cata)
    cata = np.sort(cata)
    print(data.shape)
    split_data()
    n=int(cata[-1])
    if cata[0]==0 and n!=1:
        n+=1
    print(f"n={n}")
    dim = data.shape[1]-1
    if data.shape[1]<5:
        ndGraphInitialize()
```

ndGraphInitialize():

根據資料是二維或三維來初始化訓練與測試兩張圖表，並新增圖例。

```python
#初始化三維圖表
def ndGraphInitialize():

    if data.shape[1]==4:
        print("顯示3維資料")
        ax3.cla()
        dots=ax3.scatter(train_data[:,0],train_data[:,1],train_data[:,2],c = train_data[:,3],vmin=-1,vmax=cata[-1])
        ax3.set_xlim(-20, 20)
        ax3.set_ylim(-20, 20)
        ax3.set_zlim(-20, 20)
        ax4.cla()
        ax4.scatter(test_data[:,0],test_data[:,1],test_data[:,2])
        ax4.set_xlim(-20, 20)
        ax4.set_ylim(-20, 20)
        ax4.set_zlim(-20, 20)
    elif data.shape[1]==3:
        ax3.cla()
        dots=ax3.scatter(train_data[:,0],train_data[:,1],c = train_data[:,2],vmin=-1,vmax=cata[-1])
        ax4.cla()
        ax4.scatter(test_data[:,0],test_data[:,1])

    # 創建對應 class 的 Patch，用來顯示 legend
    patchs = []
    '''
    patchs.append(Patch(color=dots.cmap(dots.norm(int(cata[-1]))), label=f'Class undefine prediction)'))
    if cata[0]==0:
        patchs.append(Patch(color=dots.cmap(dots.norm(int(cata[0]))), label=f'Class {int(cata[0])}'))
    '''
    for i in cata:
        patchs.append(Patch(color=dots.cmap(dots.norm(int(i))), label=f'Class {int(i)}'))
    # 添加 legend，並將 class 0 和 class 1 對應顏色添加到圖例中
    ax3.legend(handles=patchs)
    ax4.legend(handles=patchs)
    canvas3.draw()
    canvas3.flush_events()
    canvas4.draw()
    canvas4.flush_events()
    time.sleep(0.1)
```

ndTrain():

由介面中 advance action 下的 ” 多維/多群訓練” Button 觸發，為主要進行訓練的 function。感知機部分的實作與基礎題的 training function 大致相同，主要的差別在 w 初始化時會使用 dim+1(dimention)、將多個感知機針對一筆資料的預測結果存成 code 矩陣，用每一位代表一個種類來進行分類(ex:1000 代表 class1、0100 代表 class2、0010 代表 class3、0001 代表 class4)藉此達成多群的分類。

w 初始化時會使用 dim+1(dimention)藉此可以實現加分題 b. 能夠處理多維資料(四維以上):

```
np.random.seed()
w =[]
for k in range(n):
    w.append(np.random.rand(dim+1) * 0.01)
#w = np.insert(w,0,-1)
print(f"randomw = {w}")
#w_update(w,line,canvas)
if data.shape[1]<5:
    ndGraphInitialize()
```

將多個感知機針對一筆資料的預測結果存成 code 矩陣藉此可實現加分題 d. 可以辨識兩群以上的資料:

```
code = []
for k in range(n):
    code.append(np.dot(w[k],t_data[0:-1]))
    print(f"v{k} = {code[-1]}")
code=list(map(lambda v:1 if v>0 else 0 if v<0 else 0,code))
```

判斷正確與否並修正 w 部分(後方為圖表更新):

```
if code !=Ans:
    correct -= 1
    print("修改w")
    for k in range(n):
        if code[k]!=Ans[k]:
            rate = learningRate if code[k] < Ans[k] else -learningRate
            w[k] = w[k] + rate*t_data[0:-1]
            print(f"rate = {rate}, {learningRate*t_data[0:-1]}")
            print(f"w{k}+ {rate*t_data[0:-1]} , new w{k} = {w[k]}")
            if dim ==3:
                if len(surface)>k:
                    surface[k].remove()
                    surface[k] = nd_surface(w[k],ax3)
                else:
                    surface.append(nd_surface(w[k],ax3))
                canvas3.draw()
                canvas3.flush_events()
                time.sleep(0.1)
            elif dim==2:
                while len(surface)<=k:
                    surface.append(nw_update(w[k],ax3,-1,canvas3))
                if len(surface)>k:
                    nw_update(w[k],ax3,surface[k],canvas3)
    #w_update(w,line,canvas)
```

在修正 w 值時則根據 code 中哪一位的結果不正確就對該 w 進行修正(ex: code = [0 ,1, 1, 0],Ans =[0, 1, 0, 0]時就對 code[2]的 w 值也就是 w2 進行修改)透過 surface 矩陣紀錄與更新當前多個鍵結值所代表的多個平面或線段(透過對 ax.plot_surface 物件進行.remove()並重新繪製來達成更新面的效果)。

nw_update(w, ax, line, canvas):

這個 funciton 用於進階題中，二維資料的線段更新(於三維空間中繪製或更新二維線段)。

```python
#三維更新圖表預測線段
def nw_update(w,ax,line,canvas):
    print(w)
    c,a,b = w
    W.set(f"w:\n{float(c):.4f}\n{a:.4}\n{b:.4}\n")

    c=c*-1
    # 設定 x 的範圍，例如從 -10 到 10
    x_vals = np.linspace(-10, 10, 2)
    x_vals = x_vals
    # 計算 y 的對應值
    y_vals = - (a / b) * x_vals - (c / b)
    if line == -1:
        newLine, = ax.plot([0,0],[0,0])
        return newLine
    else:
        line.set_data_3d(x_vals,y_vals,[0,0])
    #plt.plot(x_vals, y_vals, label=f'{a}x + {b}y + {c} = 0')
    canvas.draw()
    canvas.flush_events()
    time.sleep(0.1)
```

nd_surface(w,ax):

用於繪製三維資料訓練時，w 鍵結值所代表的平面。

```python
#三維更新圖表預測面
def nd_surface(w,ax):
    d ,a, b, c = w  # 法向量
    # 定義 x 和 y 的範圍
    x = np.linspace(-10, 10, 2)
    y = np.linspace(-10, 10, 2)
    x, y = np.meshgrid(x, y)

    # 計算對應的 z 值，基於平面方程式 ax + by + cz = d
    z = (d - a * x - b * y) / c

    # 繪製平面
    return ax.plot_surface(x, y, z, alpha=0.5, rstride=100, cstride=100)
```

ndTest():

由介面中 advance action 下的 ” 多維/多群測試” Button 觸發，原理與基礎題的 testing function 大致相同，會先對所有測試資料透過內積算出預測結果

code 儲存至 predictions 矩陣中。

```
predictions=[]
for i,t_data in enumerate(test_data):
    t_data = np.insert(t_data,0,-1)
    code = []
    for k in range(n):
        code.append(np.dot(w[k],t_data[0:-1]))
        #print(f"v{k} = {code[-1]}")
    code=list(map(lambda v:1 if v>0 else 0 if v<0 else 0,code))
    sum = 0
    prediction = -1
    for k in range(n):
        sum+=code[k]
        if code[k]==1:
            prediction=k if cata[0] == 0 else k+1
    if sum>1:
        prediction=-1
    if n==1:
        prediction=code[0]
    predictions.append(prediction)
```

再判別預測正確的 index 有哪些(correct_index)，計算正確率並將正確的預測與錯誤的預測用預測出來的類別作為顏色，分別用不同的符號”o”與”x”繪製至圖表上。

```
ax4.cla()
ax4.scatter(test_data[correct_index,0],test_data[correct_index,1],test_data[correct_index,2],c=predictions[correct_index],marker='o',vmin=-1,vmax=cata[-1])
incorrect_predictions = np.delete(predictions,correct_index,axis=0)
x = test_data[:,0]
y = test_data[:,1]
z = test_data[:,2]
ax4.scatter(np.delete(x,correct_index,axis=0),np.delete(y,correct_index,axis=0),np.delete(z,correct_index,axis=0),c=incorrect_predictions,marker='x',vmin=-1,vmax=c
ax4.set_xlim(-20, 20)
ax4.set_ylim(-20, 20)
ax4.set_zlim(-20, 20)
canvas4.draw()
canvas4.flush_events()
time.sleep(0.1)
```

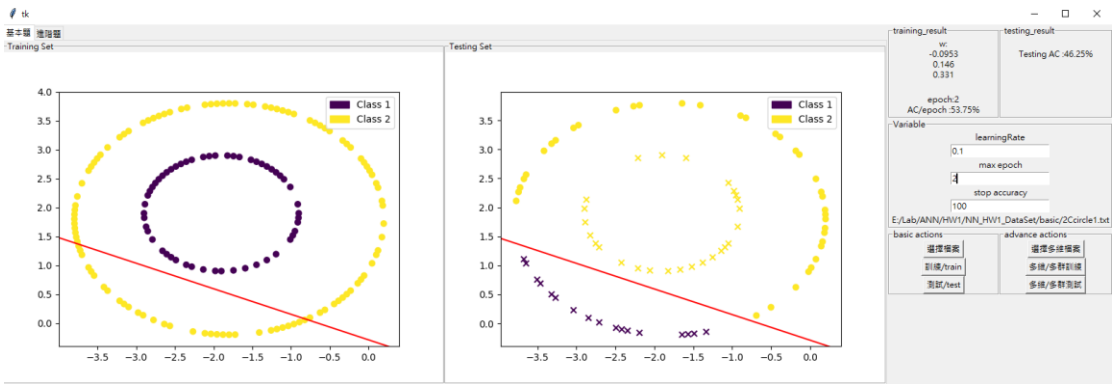
Code 轉換回 prediction class 部分的程式碼。(if n==1 的部分判別如果總共只有一個感知機則用不同的類別定義方法，非 0100 等而是單純 0 或 1)

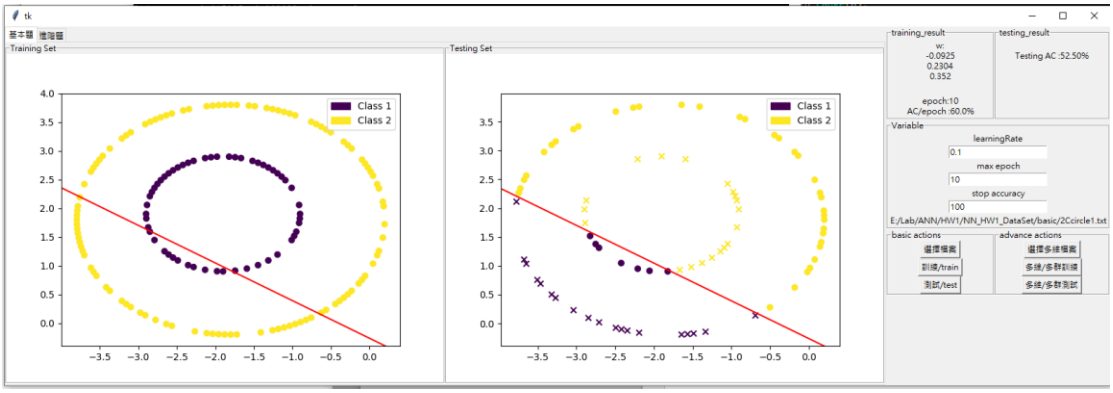
```
code = []
for k in range(n):
    code.append(np.dot(w[k],t_data[0:-1]))
    #print(f"v{k} = {code[-1]}")
code=list(map(lambda v:1 if v>0 else 0 if v<0 else 0,code))
sum = 0
prediction = -1
for k in range(n):
    sum+=code[k]
    if code[k]==1:
        prediction=k if cata[0] == 0 else k+1
if sum>1:
    prediction=-1
if n==1:
    prediction=code[0]
predictions.append(prediction)
print(f"code={code},prediction={prediction},Ans ={t_data[-1]}")
if prediction == t_data[-1]:
    correct_index.append(i)
```

三.　實驗結果

2Ccircle1

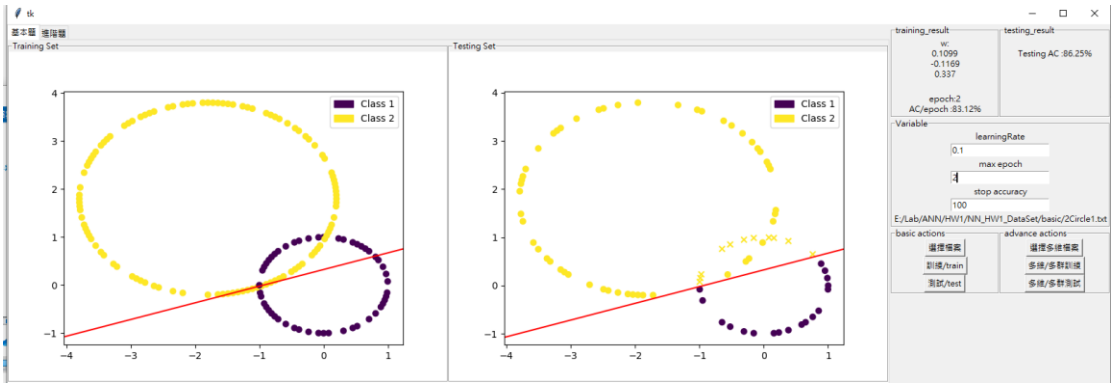
由於圖形非線性可分割，訓練 10 個 epoch 與 2 個 epoch 的結果相似，訓練準確度分別達到 53.75%與 60%，測試準確度分別為 46.25%與 52.5%。



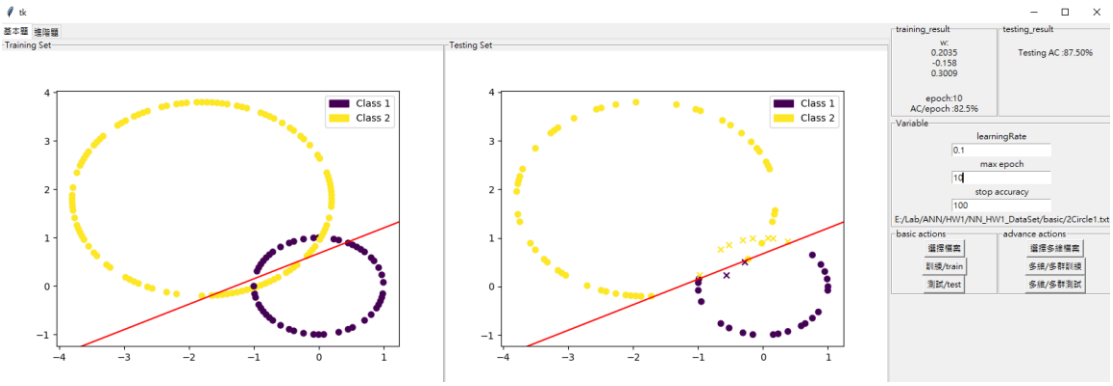


2Circle1

這題也非線性可分割問題，在訓練 2 epoch 後模型找到兩個圓交接處附近的一條線，訓練準確度為 83.12%測試準確度為 86.25%。

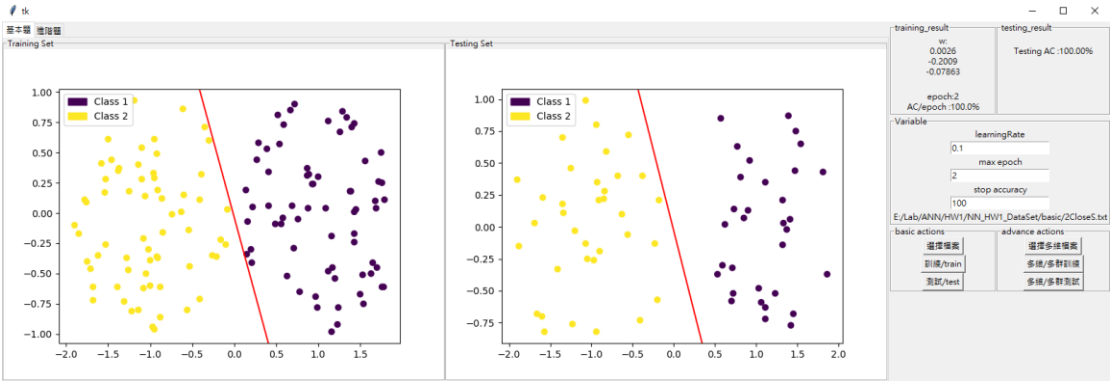


訓練 10 epoch 的訓練準確度為 82.25%測試準確度為 87.5%

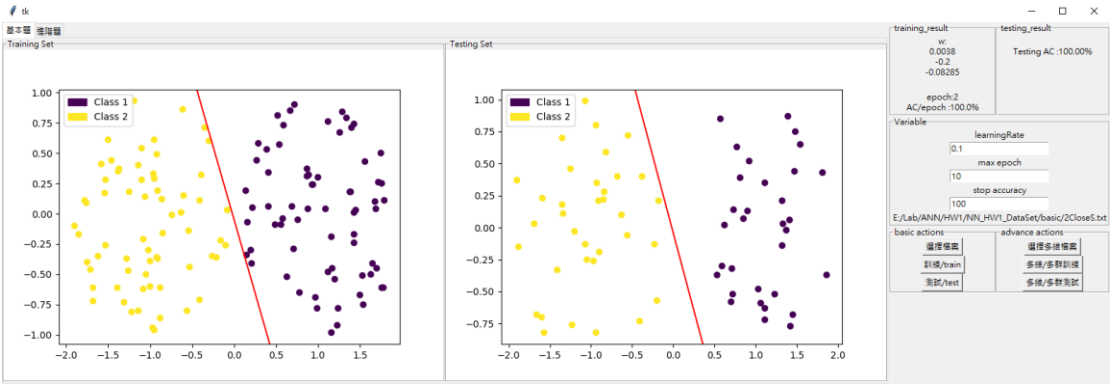


2CloseS

這題為線性可分割問題，感知機在 epoch 2 就獲得了訓練集與測試集都為 100%準確率的成果。

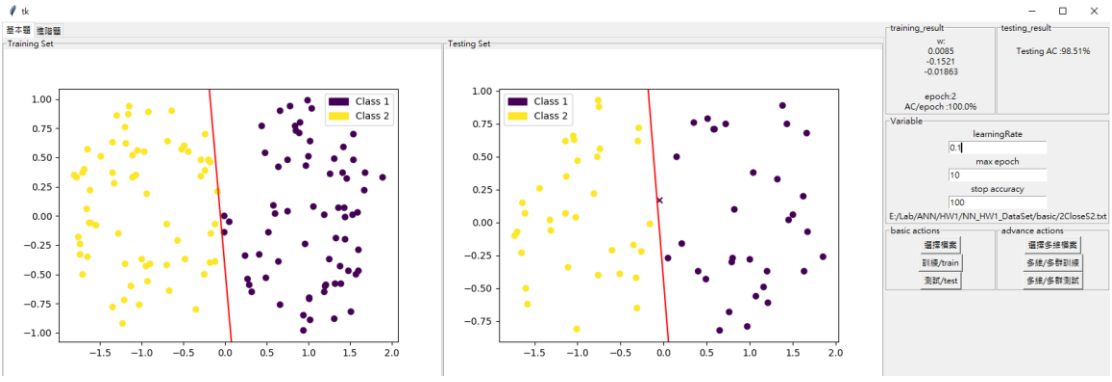


此時就算將 epoch 設為 10，程式也會在 epoch 2 時因為已經達到設定的 stop accuracy 100%停止。

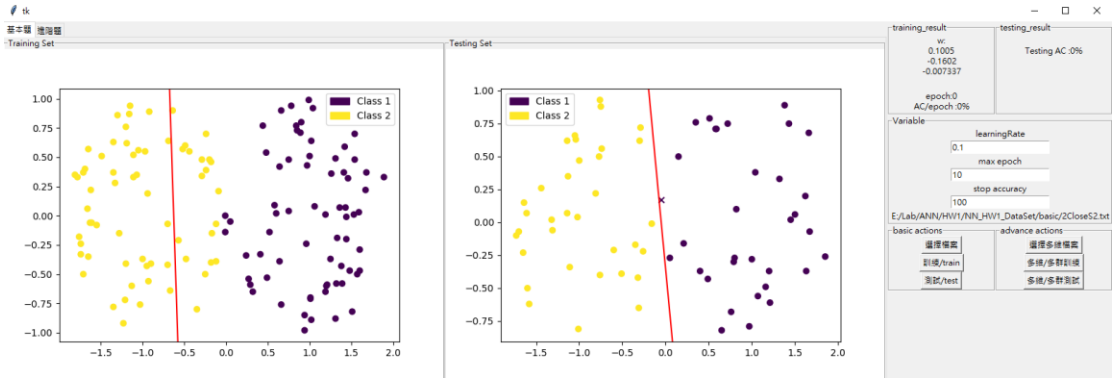


2CloseS2

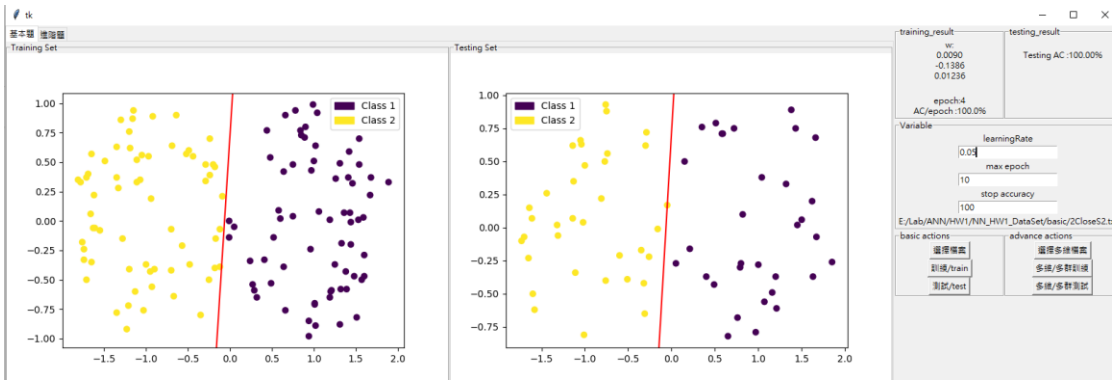
此題雖為線性可分割問題，但兩類點較為接近，第一次測試 epoch 設為 10，感知機在 epoch 2 時因達到設定的 accuracy 100%停止，但於 testing set 中得到了一個錯誤點，準確度來到 98.51%。



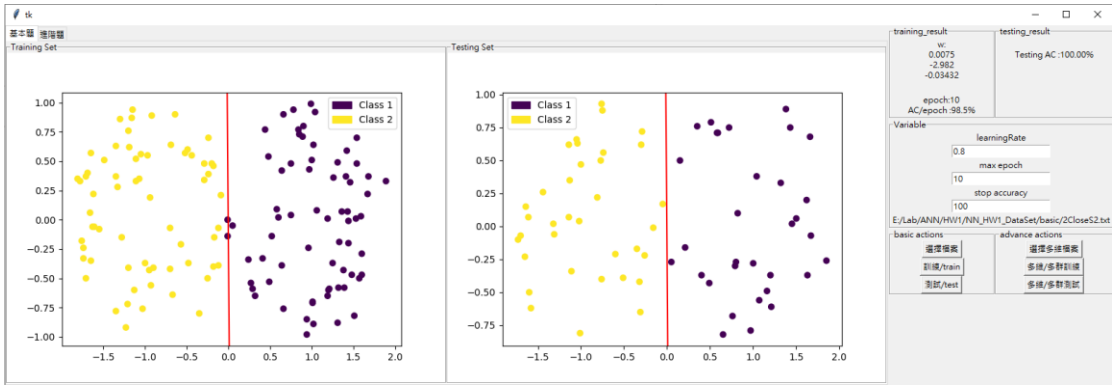
觀察訓練過程會發現線段跳躍的幅度較大(下圖)，故可將 learning rate 下降



下圖將 learning rate 下降後線段跳躍的幅度顯著減少，並於 epoch 4 時找到訓練集與測試集階為 100%準確度的結果。



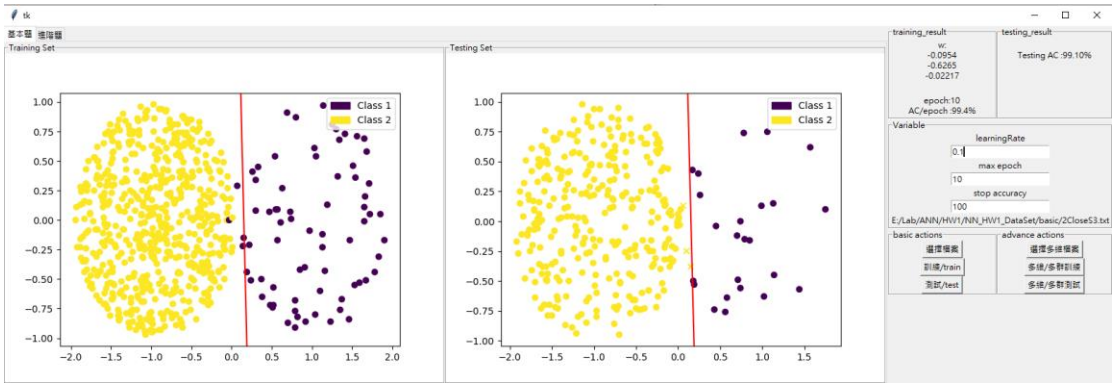
我們也可以嘗試反向將 learning rate 加大到 0.8



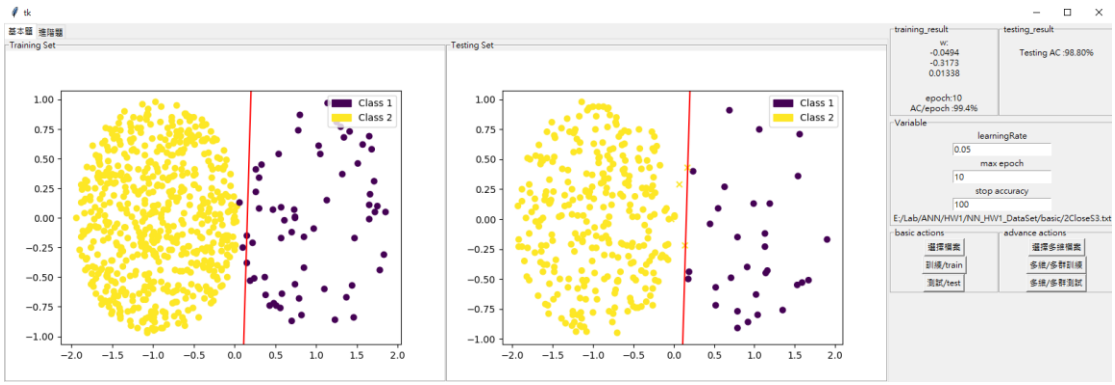
可以看到感知機無超過 epoch 10 都無法找到訓練及準確度 100%的小縫隙，這是因為每次跳動修正的幅度過大導致。

2CloseS3

這題也非線性分割問題，感知機在 epoch 10 得到訓練資料準確度 99.4%，測試準確度 99.1%的結果。

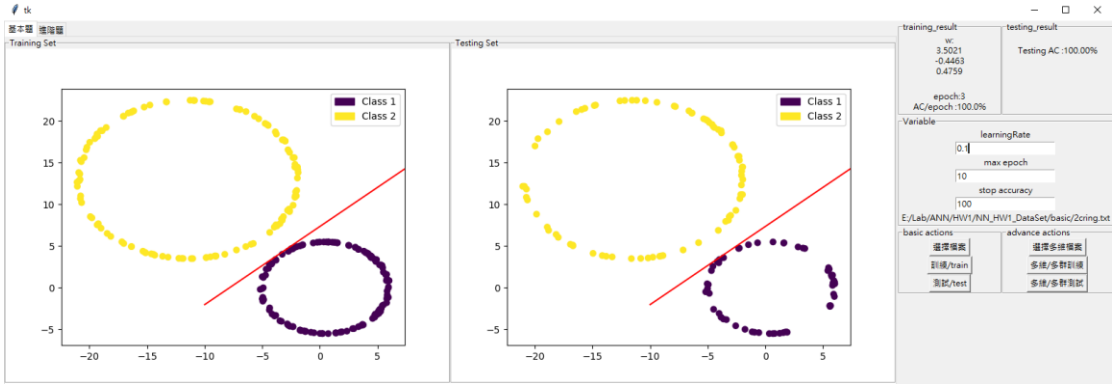


第二次測試也得到相似的結果，感知機在 epoch 10 得到訓練資料準確度 99.4%，測試準確度 98.8%



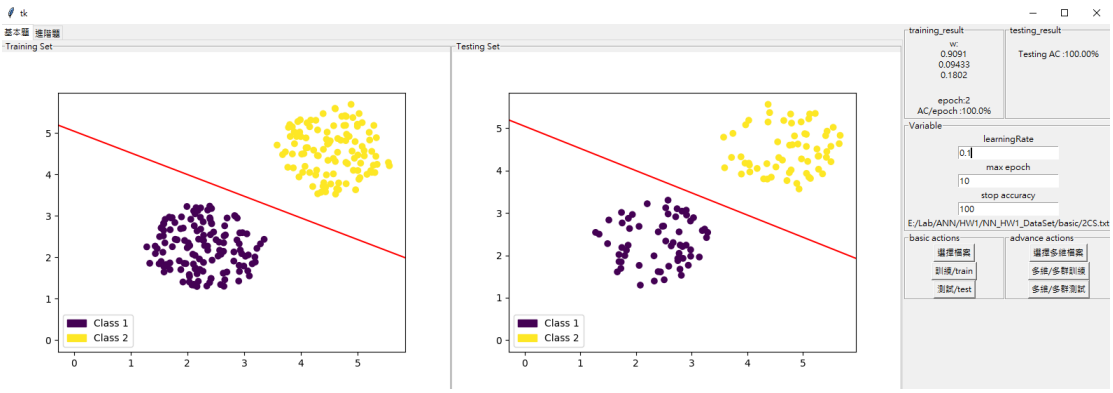
2cring

這題為線性可分割問題，感知機在 epoch 2 就獲得了訓練集與測試集都為 100% 準確率的成果。



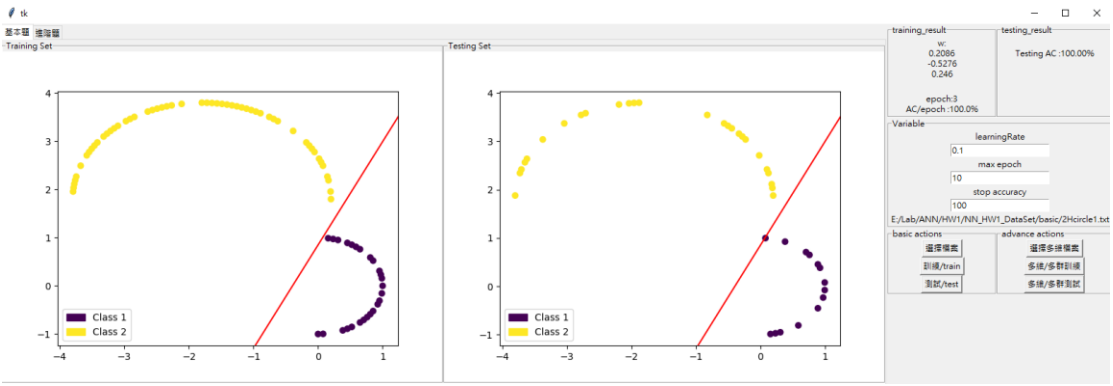
2CS

這題也為線性可分割問題，感知機與上題一樣在 epoch 2 就獲得了訓練集與測試集都為 100%準確率的成果。



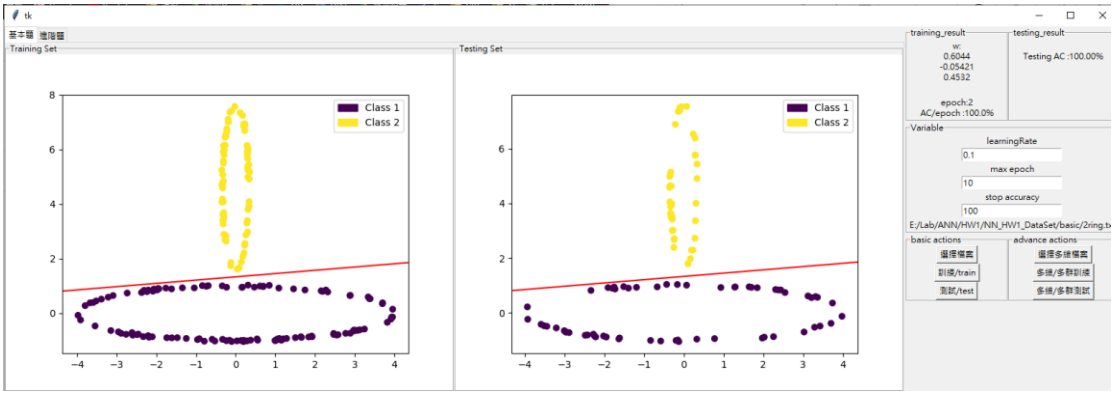
2Hcircle1

這題也為線性可分割問題，感知機與上題一樣在 epoch 3 就獲得了訓練集與測試集都為 100%準確率的成果。



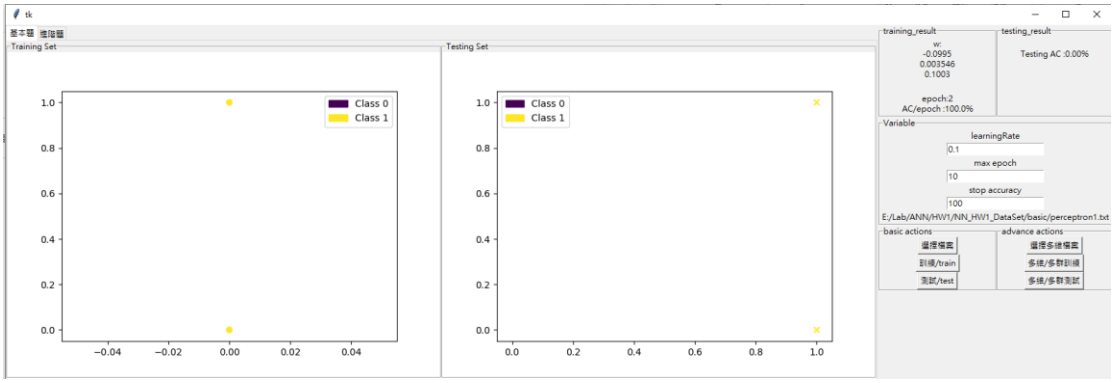
2ring

這題也為線性可分割問題，感知機與上題一樣在 epoch 2 就獲得了訓練集與測試集都為 100%準確率的成果。

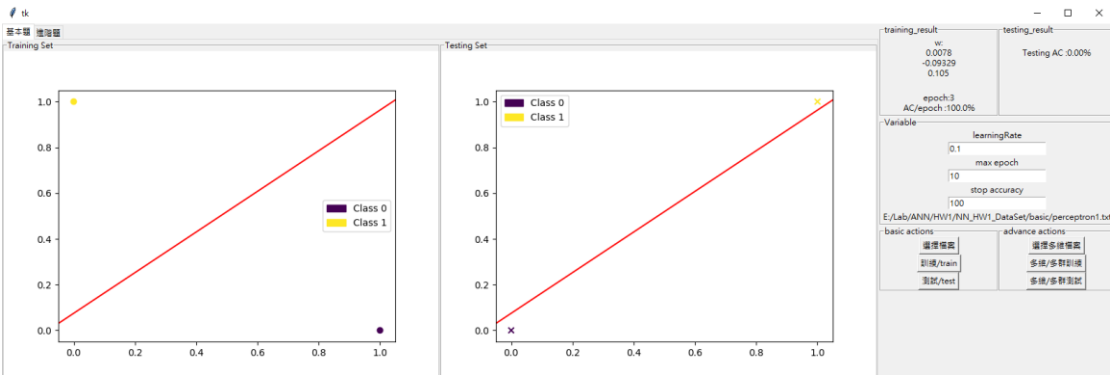


perceptron1

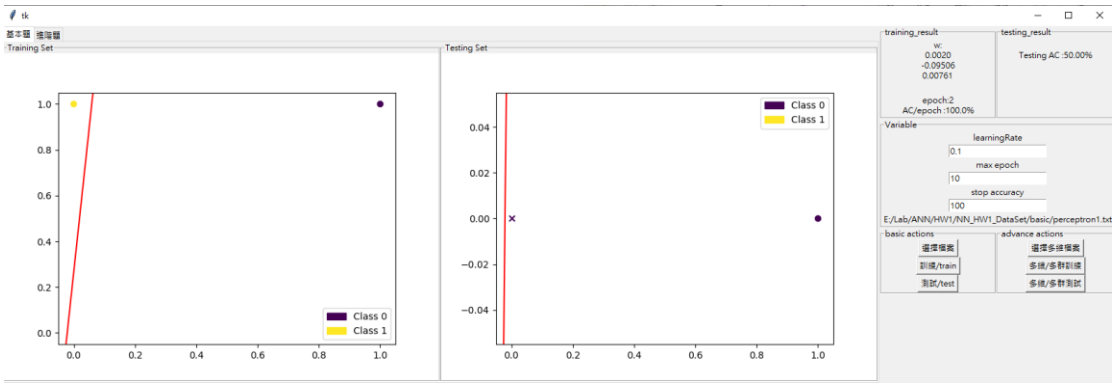
這題為線性可分割問題，但第一次測試中訓練及抽到的兩點皆為 class 1，導致感知機沒有進行到訓練，結果為訓練集 100%，測試集 0%準確度



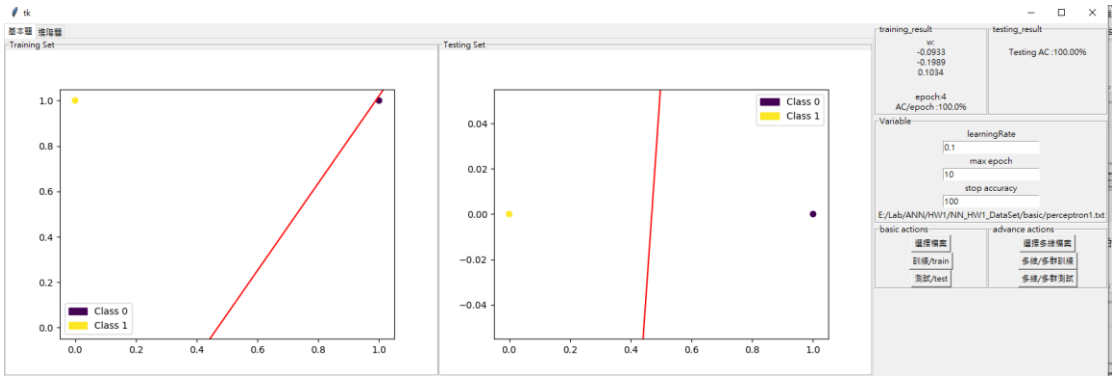
第二次測試雖抽到了兩點不同 class 可以進行訓練，但由於訓練資料的代表性不足(沒有很好的表現出母體分布特徵)導致感知機沒有找到真正的母體分布規律，訓練集 100%，測試集 0%準確度。



第三次測試，雖然訓練資料集展現了母體的主要規律，但感知機仍以一點點的誤差於測試集中只拿到 50%的準確度。

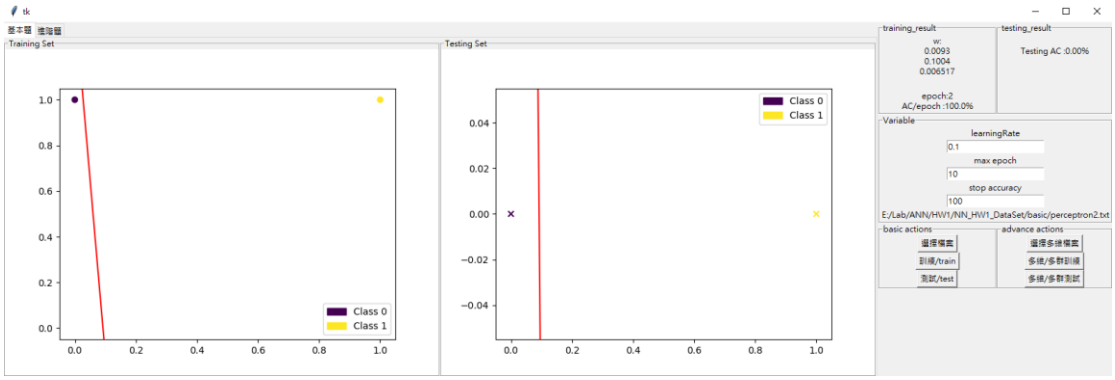


第四次測試，雖然不是最完美的分割線，但感知機成功得到了兩個 100%的結果。



perceptron2

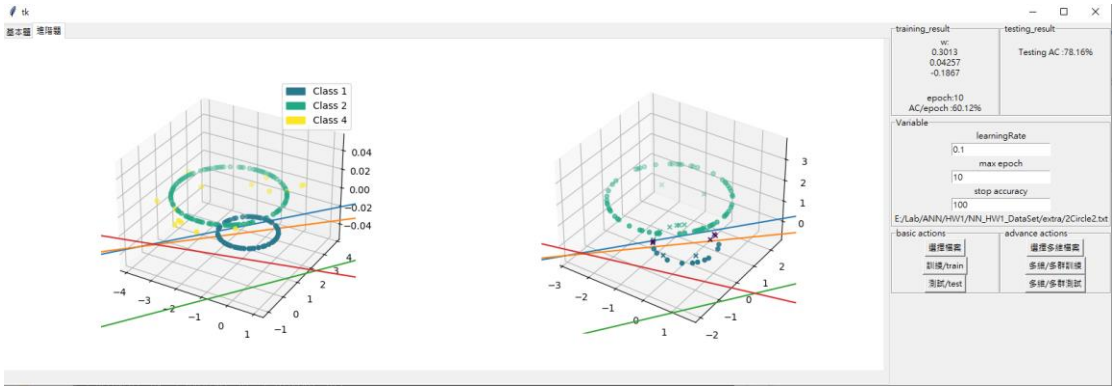
這題的母體資料分布為 XOR，故單個感知機無法達成完全正確的分類，最終得到結果訓練集 100%測試集 0%的準確度。



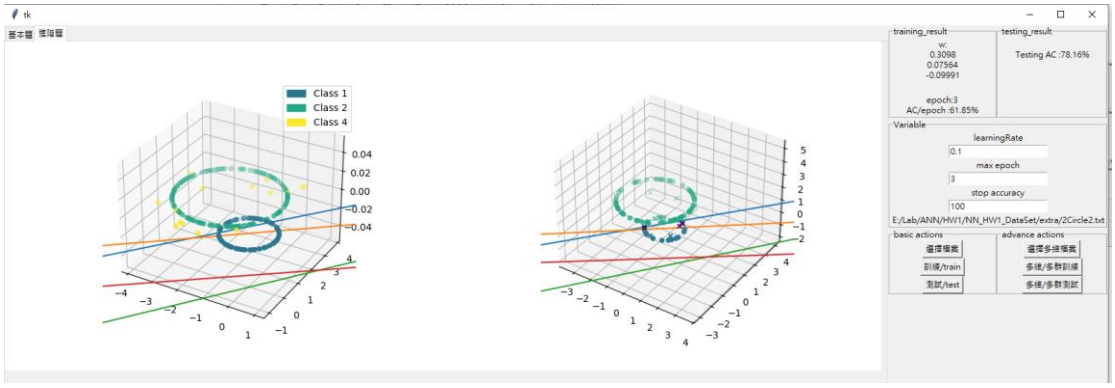
四.　進階題實驗結果

2Circle2

此題包含多群分類總共 4 種類別，但 class3 中沒有資料，程式使用了 4 條(對應 4 個 class)嘗試進行分類，其中綠色線段對應沒資料的 class3 故訓練過程中沒更新，而由於 class4 的黃點分布較為隨機，紅色線段沒能成功判別初 class4 的資料點，測試資料中的紫色點代表項是 code 0110、1010 等沒有被定義的類別，是直接使用等同類別數量的感知機所會碰到的問題，最後得到訓練 60.12%測試 78.16%的結果。



第二次測試 3 epoch 也得到相似的結果。



4satellite-6

這題為 4 維資料，總共有 6 種類別。

從 consol 中可以觀察訓練與修正過程:

```
v0 = -5206.513513768683
v1 = 39.12905406540017
v2 = -3506.01510377708
v3 = -2089.2188679732367
v4 = -956.2095191237104
v5 = 1427.243694602933
[-1. 68. 69. 86. 72.  5.]
prediction = [0, 1, 0, 0, 0, 1] , Answer = [0, 0, 0, 0, 1, 0] , w = [array([-8.59758168e+00, -3.79893589e+02,  2.5460219
1e+02,  3.01751437e-01,
        4.20020991e+01]), array([ 6.40020471e+00,  8.78112004e-05, -9.01908822e+01,  5.74023900e+01,
        1.85012308e+01]), array([  65.00838006,   20.80780652,  153.00881215, -113.09262165,
        -78.99416866]), array([  7.80664571,  35.80232419,  46.30025874, -27.29087253,
       -74.49545948]), array([-32.79662665, 125.40097075, -97.7907294 ,  19.70571021,
       -61.99170703]), array([ -28.29080517,  203.90614711,  -12.39348589, -100.19311902,
        -41.59592156])]
修改w
rate = -0.1, [-0.1  6.8  6.9  8.6  7.2]
w1+ [ 0.1 -6.8 -6.9 -8.6 -7.2] , new w1 = [  6.50020471  -6.79991219 -97.09088223  48.80239002  11.30123083]
rate = 0.1, [-0.1  6.8  6.9  8.6  7.2]
w4+ [-0.1  6.8  6.9  8.6  7.2] , new w4 = [-32.89662665 132.20097075 -90.8907294   28.30571021 -54.79170703]
rate = -0.1, [-0.1  6.8  6.9  8.6  7.2]
w5+ [ 0.1 -6.8 -6.9 -8.6 -7.2] , new w5 = [ -28.19080517  197.10614711  -19.29348589 -108.79311902  -48.79592156]
```

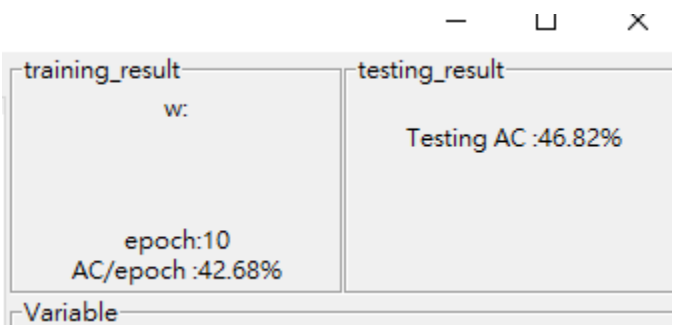
上圖中為一筆資料的訓練與修正過程，圖中間顯示這筆資料的預測結果 prediction code 為 010001 但正確結果為 000010，故於最下方對 w1、w4、w5 進行了修正。

```
epoch: 10 訓練正確率Accuracy42.68098647573588%
[36.43595863166269, 41.84566428003182, 42.482100238663485, 43.71519490851233, 42.561654733492446, 42.60143198090692, 42.
44232299124901, 42.80031821797932, 42.720763723150355, 42.68098647573588]
```

可看到 epoch10 的最終訓練準確度為 42.68%，最下方 list 為每個訓練 epoch 的準確度，可以看到最高在 42%就難以繼續上升了。

同樣可於 consol 中觀察測試結果:

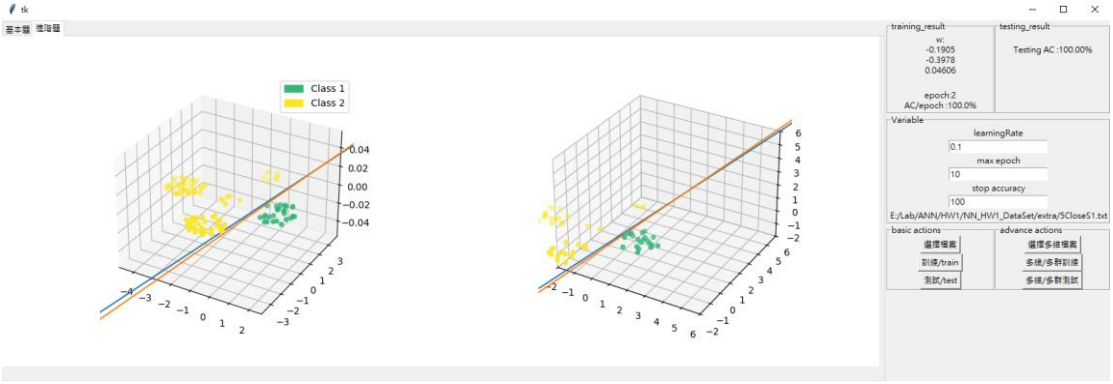
```
code=[0, 0, 1, 0, 0, 1],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 1],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 1],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 0, 0, 0, 0],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[1, 0, 0, 0, 0, 0],prediction=1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 0, 0, 0, 0],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 0, 0, 0, 0],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 1],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
code=[0, 0, 0, 0, 0, 0],prediction=-1,Ans =3.0
code=[0, 0, 1, 0, 0, 0],prediction=3,Ans =3.0
```



最終 epoch 10 達到訓練集 42.68%，測試 46.82%的結果。

5CloseS1

這題較為簡單，二維且只有兩類，程式使用了兩個感知機來預測並於 epoch2 就達成了兩個 100%的結果。

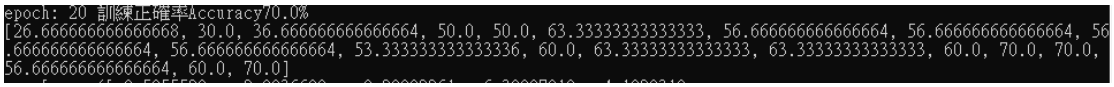


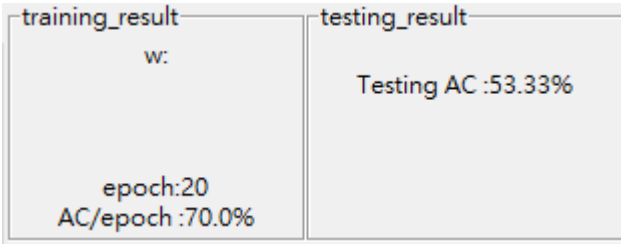
8OX

這題為 8 維資料、共 3 個 class 的分類，程式使用三個感知機。

第一次測試 10 個 epoch， 訓練準確率達到 63.33%，測試準確度為 33%

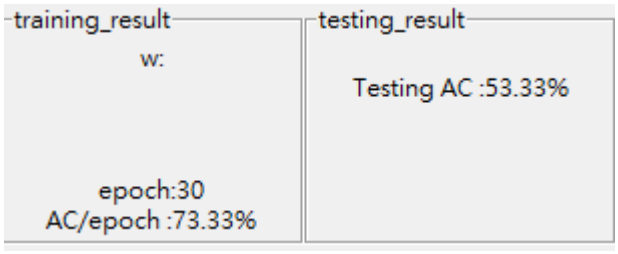
第二次測試提升 epoch 到 20，得到訓練集 70%，測試集 53.33%的結果有些微提升。





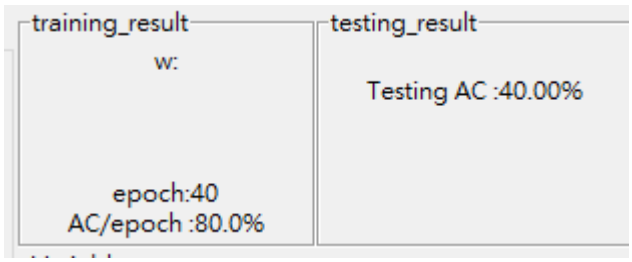
第三次測試提升 epoch 到 30，雖得到訓練集 73.33%，但測試集 53.33%的結果沒有提升。

```
epoch: 30 訓練正確率Accuracy73.33333333333333%
[26.666666666666668, 30.0, 36.666666666666664, 53.333333333333336, 46.666666666666664, 53.333333333333336, 66.666666
6666, 53.333333333333336, 63.33333333333333, 66.66666666666666, 60.0, 60.0, 66.66666666666666, 76.66666666666667, 70
66.66666666666666, 66.66666666666666, 56.666666666666664, 60.0, 80.0, 66.66666666666666, 66.66666666666666, 73.33333
33333, 70.0, 76.66666666666667, 80.0, 70.0, 70.0, 70.0, 73.33333333333333]
```



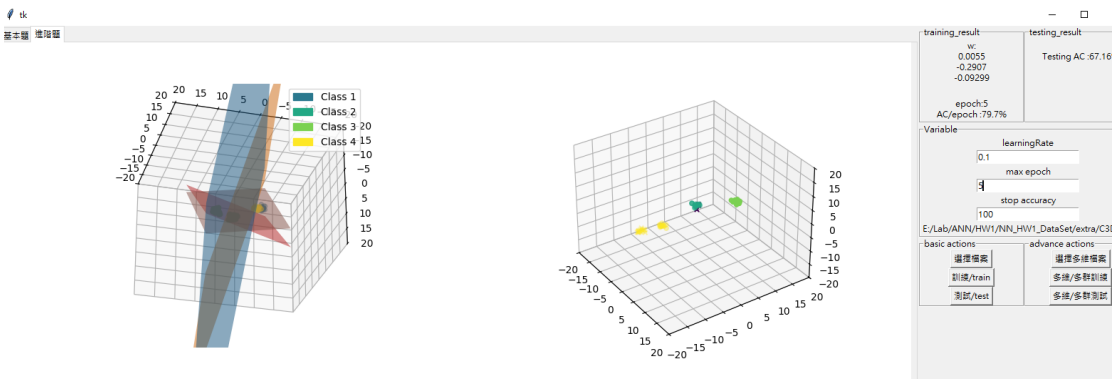
第四次測試提升 epoch 到 40，雖得到訓練集 80%的準確度看似有顯著的提升，但測試集的結果不僅沒有提升反而是大幅的下降到了剩下 40%，可見明顯出現對於訓練資料過擬合的問題了。

```
epoch: 40 訓練正確率Accuracy80.0%
[26.666666666666668, 30.0, 36.666666666666664, 50.0, 50.0, 63.33333333333333, 56.666666666666664, 56.666666666666664, 56
.666666666666664, 56.666666666666664, 53.333333333333336, 60.0, 63.33333333333333, 63.33333333333333, 60.0, 70.0, 70.0,
56.666666666666664, 60.0, 70.0, 70.0, 73.33333333333333, 63.33333333333333, 83.33333333333334, 70.0, 66.66666666666666,
70.0, 83.33333333333334, 60.0, 66.66666666666666, 70.0, 73.33333333333333, 66.66666666666666, 73.33333333333333, 73.3333
3333333333, 80.0, 90.0, 80.0, 83.33333333333334, 80.0]
```

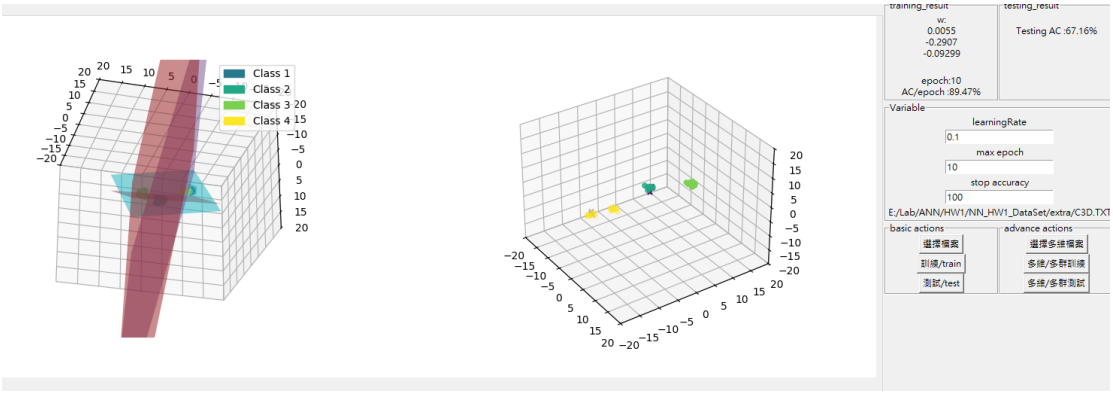


C3D

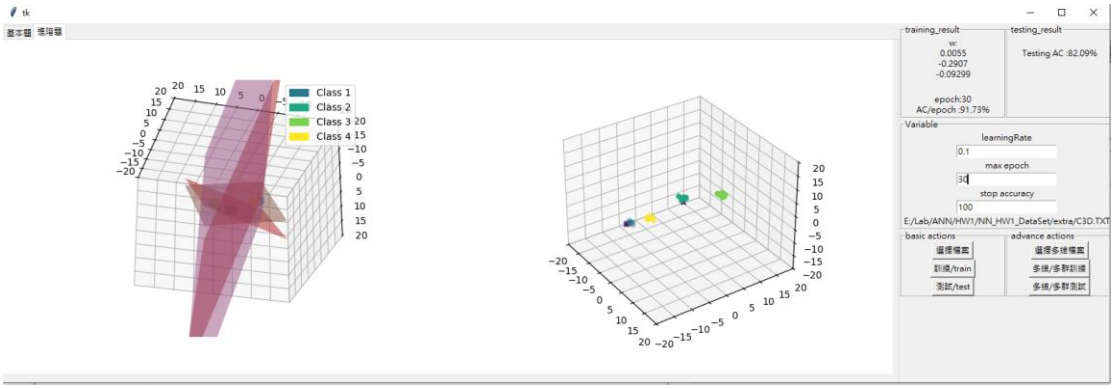
第一次測試，5 個 epoch，訓練正確率達到 79.7%測試正確率達到 67.16%



第二次測試，10 個 epoch，訓練正確率達到 89.47%測試正確率還是 67.16%



第三次測試, 共 30 個 epoch 訓練正確率達到 91.73%測試正確率達到 82.09%有顯著的提升。

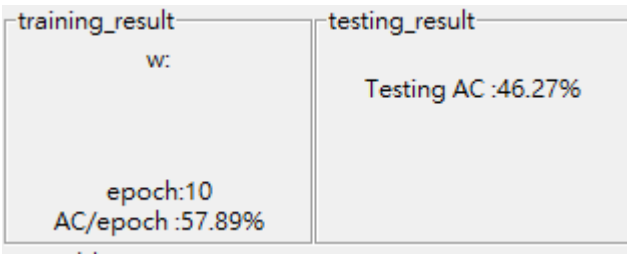


C10D

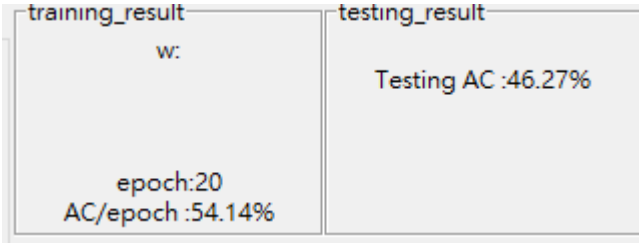
這題為 10 維共 4 class 的分類，程式使用了 4 個感知機。

第一次測試 10 個 epoch，訓練正確率達到 57.89%測試正確率達到 46.27%

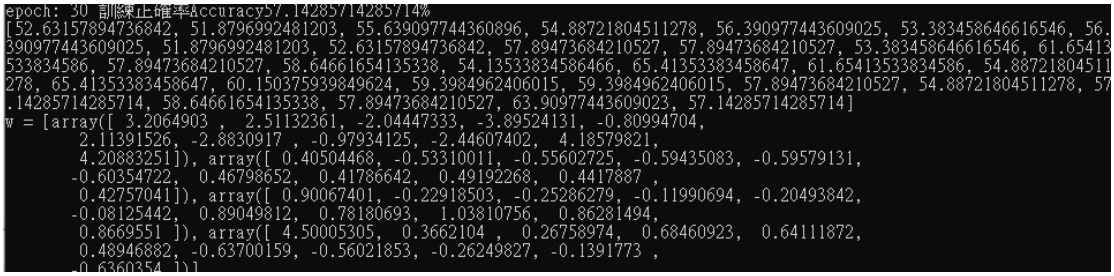
```
epoch: 10 訓練正確率Accuracy57.89473684210527%
[52.63157894736842, 53.383458646616546, 55.639097744360896, 54.13533834586466, 57.14285714285714, 52.63157894736842, 57.14285714285714, 57.14285714285714, 54.13533834586466, 57.89473684210527]
w = [array([ 2.30800364,  0.42849907, -1.08729295, -2.10714265, -0.63679786,
       -0.31356138, -0.95928129, -0.68959308, -1.09672321,  1.89123249,
        1.91987232]), array([ 0.5060937 , -0.55738086, -0.42497356, -0.44990877, -0.38476929,
       -0.51413685, -0.39058113, -0.43920161, -0.42344765, -0.3774169 ,
       -0.50895675]), array([ 0.9063278 , -0.1283117 , -0.20039694, -0.11321554, -0.17453575,
       -0.12223695,  0.83346652,  0.73564272,  1.03268939,  0.84827976,
        0.85400819]), array([ 1.20966023,  0.4831244 ,  0.46830152,  0.64426194,  0.46055667,
        0.53994788, -0.6491321 , -0.54097228, -0.46693287, -0.37741249,
       -0.5649298 ])]
```

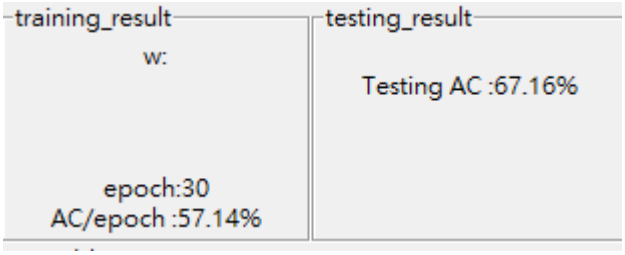


第二次測試提升 epoch 至 20，結果沒有提升。



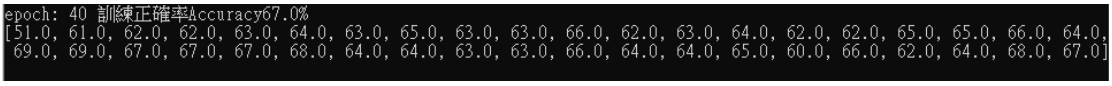
第三測試提升 epoch 至 30，測試集的結果顯著提升至 67.16%。

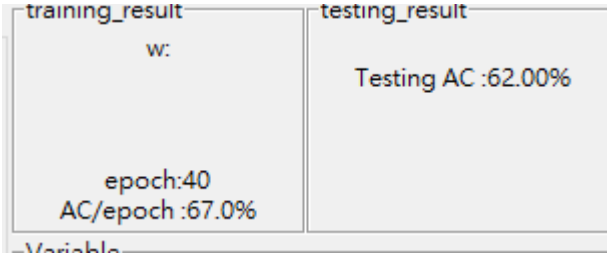




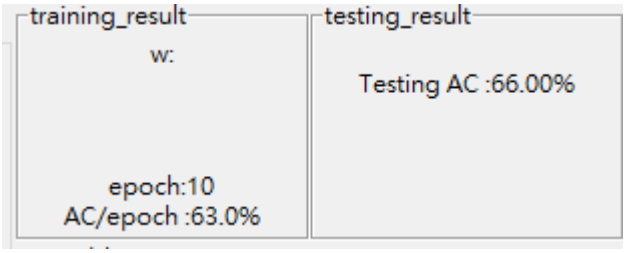
IRIS

第一次測試 40 epoch，訓練集準確度 67%測試集準確度 62%

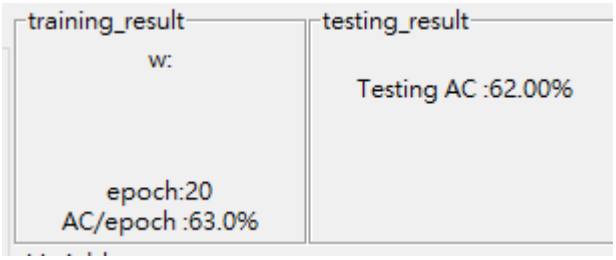




第二測試 10 epoch，訓練集準確度 63%測試集準確度 66%

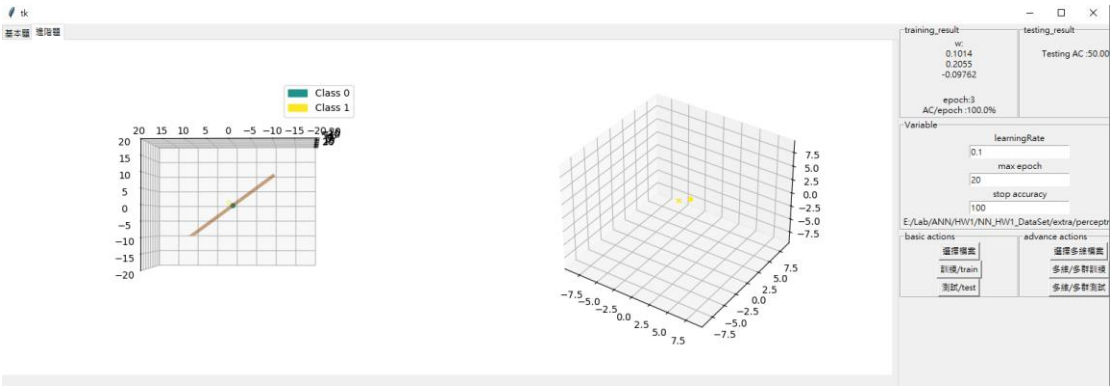


另外分別測試 20 epoch 與 30 epoch 結果沒有顯著差別

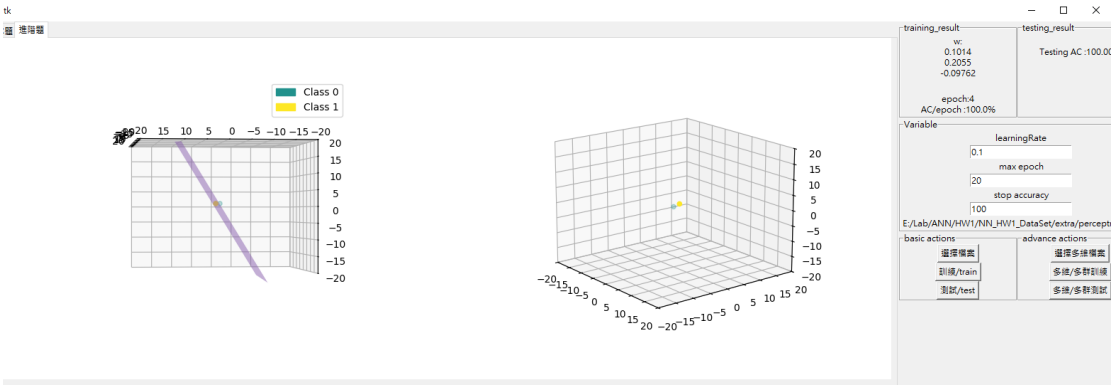


perceptron3

第一次測試，模型在 epoch3 時得到訓練集 100%測試集 50%的結果



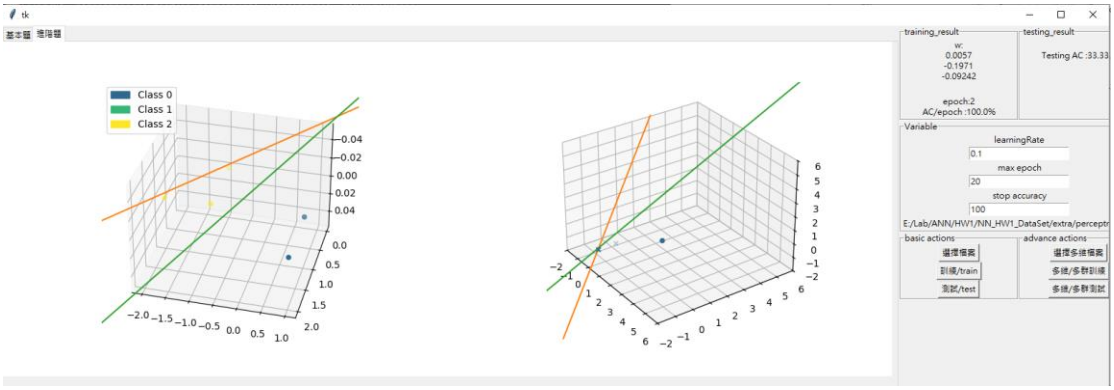
第二次測試，模型在 epoch3 時得到訓練集 100%測試集 100%的結果



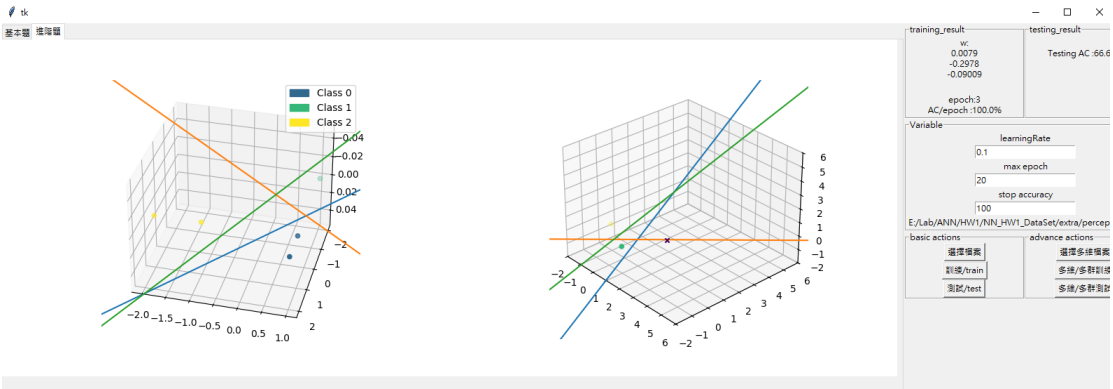
個人認為因為訓練資料太少不足以有效表現出所有母體特徵，以至於有點碰運氣切割的感覺在。

perceptron4

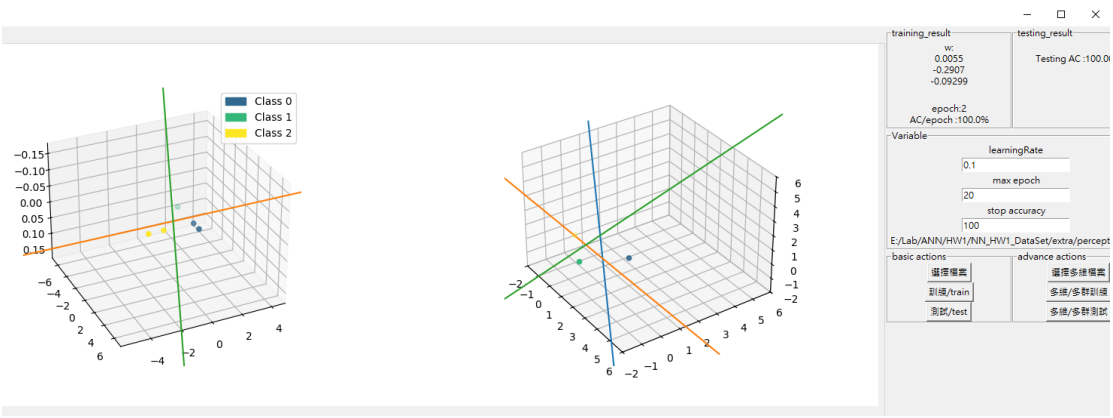
第一次測試:訓練集中沒有選到 class 二的資料點以至於第三條線沒有正常訓練到，整體效過並不好雖訓練集準確度 100%但測試集準確率只有 33%。



第二次測試於 epoch 3 得到訓練集準確度 100%測試集準確率 66%效果還不錯。



第三次測試: 於 epoch 3 得到訓練集準確度 100%測試集準確率 100%

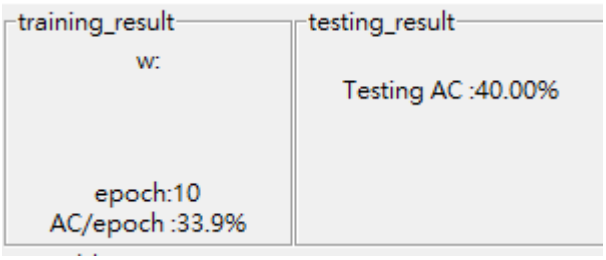


wine

共 13 維 3 種 class

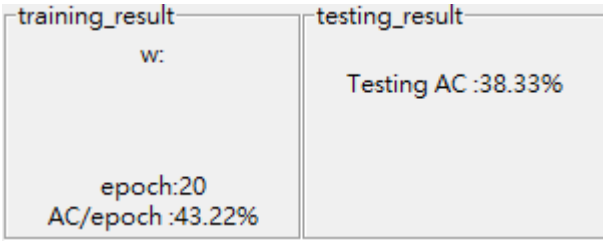
第一次測試: 10 epoch 訓練集準確度 33.9%測試集準確度 40%

```
epoch: 10 訓練正確率Accuracy33.898305084745776%
[27.11864406779661, 25.423728813559322, 33.89830508474576, 29.66101694915254, 33.05084745762712, 38.13559322033898, 3
4576271186441, 36.440677966101696, 38.983050847457626, 33.89830508474576]
```



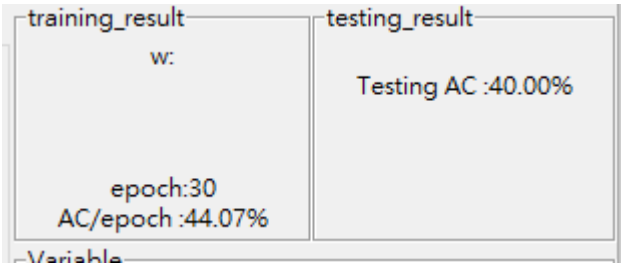
第二次測試: 20 epoch 訓練集準確度 43.2%測試集準確度 38.33%

```
epoch: 20 訓練正確率Accuracy43.220338983050850%
[27.11864406779661, 25.423728813559322, 33.89830508474576, 29.66101694915254, 33.05084745762712, 38.13559322033898, 3
4576271186441, 36.440677966101696, 38.983050847457626, 33.89830508474576, 39.83050847457627, 38.13559322033898, 38.13
322033898, 36.440677966101696, 41.52542372881356, 43.22033898305085, 42.3728813559322, 42.3728813559322, 42.372881355
2, 43.22033898305085]
```



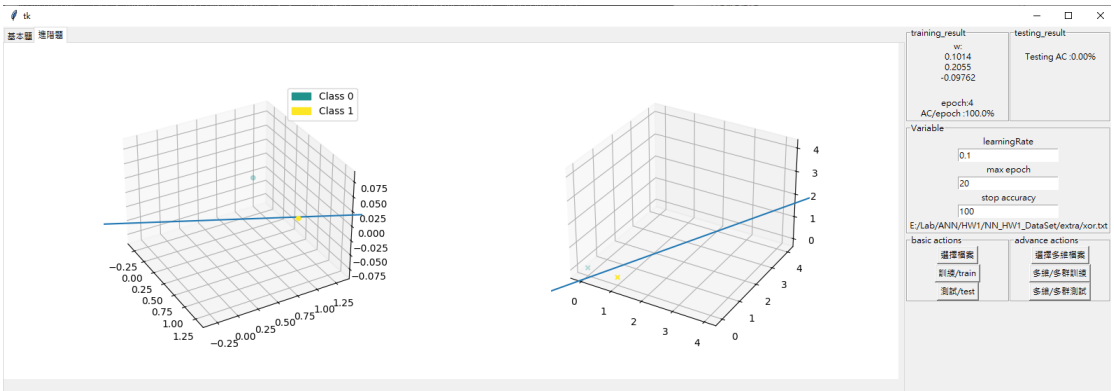
第三次測試:30 epoch 訓練集準確度 44.07%測試集準確度 40%。沒有提升

```
epoch: 30 訓練正確率accuracy44.06779661016949%
[27.11864406779661, 25.423728813559322, 33.89830508474576, 29.66101694915254, 33.05084745762712, 38.13559322033898, 34.74576271186441, 36.440677966101696, 38.983050847457626, 33.89830508474576, 39.83050847457627, 38.13559322033898, 38.13559322033898, 36.440677966101696, 41.52542372881356, 43.22033898305085, 42.3728813559322, 42.3728813559322, 42.37288135593222, 43.22033898305085, 42.3728813559322, 45.76271186440678, 44.06779661016949, 39.83050847457627, 43.22033898305085, 40.67796610169492, 39.83050847457627, 45.76271186440678, 49.152542372881356, 44.06779661016949]
```



xor

這題 XOR 需要多個感知機才能正確分類出所有點，程式使用 1 個感知機故無法正確分類，得到了訓練集 100%測試集 0%的結果。



五. 實驗結果總結。

這次實驗中觀察到了以下現象

- 透過圖形化介面觀察了線性可分割與非線性可分割兩種類型對於感知機切割時的效果。
- 透過基本題中線性可分割的 2CloseS2，觀察測試了過大 learning rate 可能造成每次修正過大而無法找到真正最優解的問題。
- 於基礎題 perceptron1、加分題的 perceptron3、perceptron4 等題目觀察到，當資料太少或品質不佳時可能會由於訓練資料的代表性不足(訓練資料沒有很好的表現出母體分佈特徵)導致感知機無法找到真正的母體分布規律

的問題 ex:(在上述這幾題中多次碰到訓練集的所有點皆維同類，或抽到的兩點沒有表現出特殊排列規則)。

- 於進階題的 80X 題目觀察了過擬合(提升 epoch 訓練集準確度提升但訓練集準確度顯著下降的狀況)。