Part 2. Programming assignment : (70%)

1. (30 points) Describe your model architecture details and PCA method.

這次作業總共做了兩個 model，分別是兩層的 NN 和三層 NN。
架構大致如下：

```
class NN(self):                    #建立 NN 框架
    def __init__(self):            #初始化
        設定輸入層、輸出層和隱藏層的 node 數
        設定 random seed 以便重現結果
        給定初始隨機權重 (w1、w2)
        給定初始 bais (b1、b2)

    def relu(self, X):             #建立激勵函數 relu function

    def _d_relu(self, X):          #建立激勵函數 relu 的導數

    def softmax(self, z):          #建立 softmax function
                                   #output 前使用，將數值換成
                                   機率輸出
    def forward_pass(self, X):     #feedforward
        z = 各層訓練資料(X 或 a)乘上權重(w)再加上 bias(b)
        if 在中間層:
            a = relu(z)            #進入激勵函數後得到的值
        elif 在 output 前的層:
            a = softmax(z)

    def backward_pass(self, y_pre, X, y): #backpropagation
        delta = 預測值(y_pre)
        權重的變化(dw) = 訓練資料(X 或 a)乘上 delta
        Bias 的變化(db) = sum delta 或 sum dz
                  (dz) = delta 乘上 w 再乘上_d_relu(a)
```

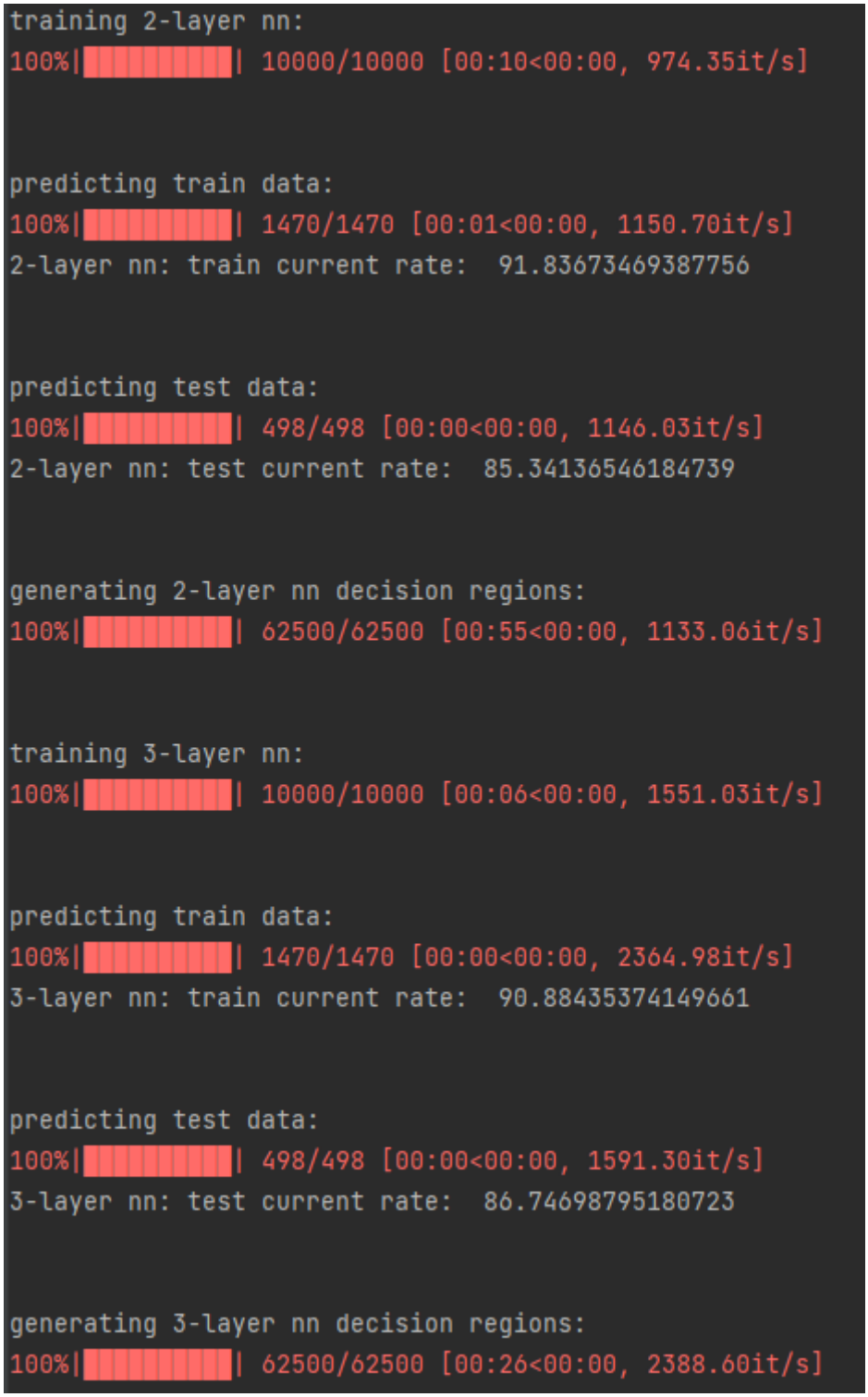
```
def loss(self, y_hat, y):   #cross_entropy_loss

def _update(self, learning_rate):   #更新權重和 bias
    w = w - learning_rate 乘上 dw
    b = b - learning_rate 乘上 db

def train(self, X, iteration):      #訓練模型
    for i in range(iteration):
        以 i 隨機取樣資料最為訓練 (SGD，沒有使用 batch)
        y_hat = forward_pass(X)
        loss = loss(y_hat, X)
        backward_pass(y_hat, X(x1, x2), X(label))
        _update()
```

在兩層的 NN，隱藏層 node 數使用 1024 個；三層的 NN 因為運算速度的關係，是使用 256 和 128 個，且兩者的 seed 都是設定為 80，learning rate = 0.01，初始隨機權重則是統一乘上 0.2，若乘上較大的值，可能導致溢位。

至於 PCA 的部分，在 training data 是使用 pca.fit_transform，提取 490*3 筆 training data 中最重要的兩個 feature，而在 testing data 則改為使用 pca.transform，提取 166*3 筆資料中與 training data 提取相符的兩個 feature。

一開始實作時沒有注意到這個細節，兩者都是使用 pca.fit_transform，所以在預測 testing data 時，不管怎麼做，正確率都很低，且在 decision regions 中會看到，預測的範圍和正確的資料點位相去甚遠，後來是在助教的 tutorial code 中發現這個問題，正確率才得以大幅提高。

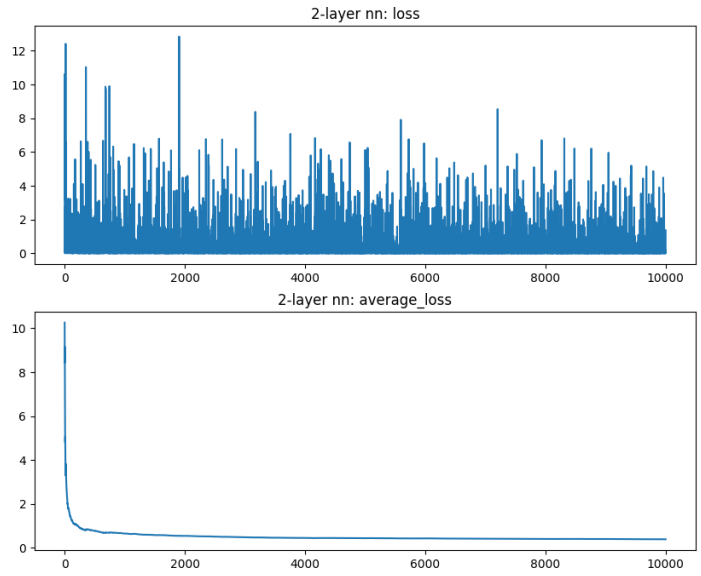
2. (10 points) Show your test accuracy.



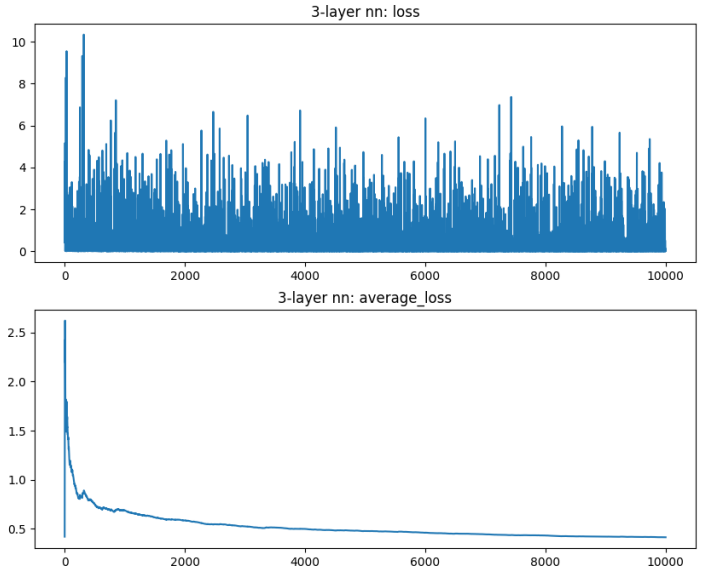
(圖1)執行訓練與預測結果 程式截圖

在兩層NN中，training data的準確率有91%，testing data則是85%；三層NN中，training data的準確率有90%，testing data則是86%。

3. (10 points) Plot training loss curves.



（圖2）兩層NN的loss與平均loss

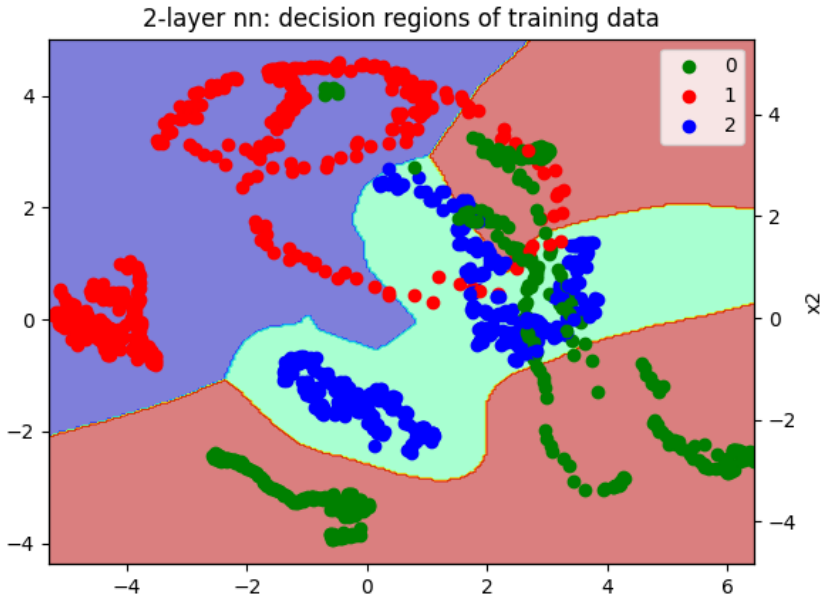


（圖3）三層NN的loss與平均loss

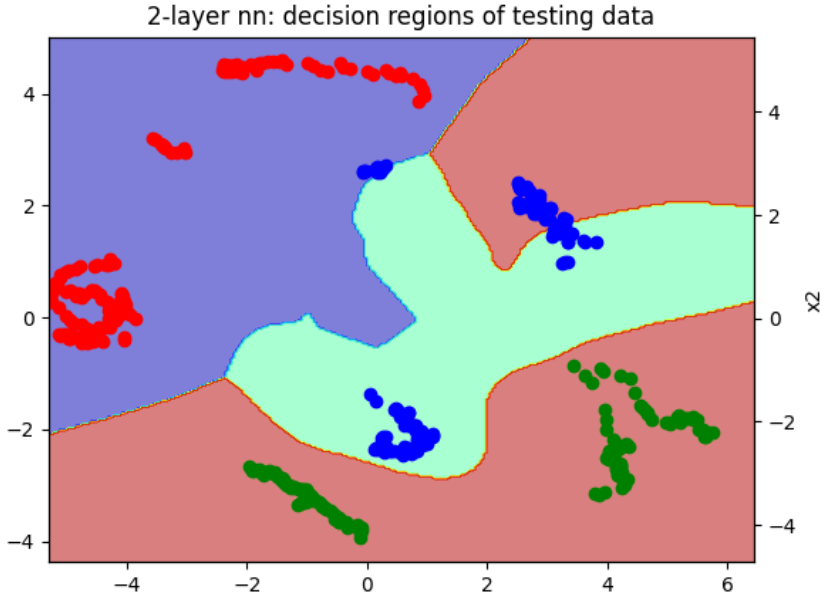
以上兩圖分別為兩層NN和三層NN的loss curve，而個別圖中的上subplot為loss值，可以發現到其波動很大，可能是SGD的隨機取樣導致，因為一次只取一筆資料，導致兩次更新間的梯度差距過大，若改用mini-batch SGD取樣，應該可以較為穩定；個別圖中的下subplot為loss的平均值，可以看出loss下降的趨勢。

4. (20 points) Plot decision regions and discuss the training/testing performance with different settings designed by yourself.

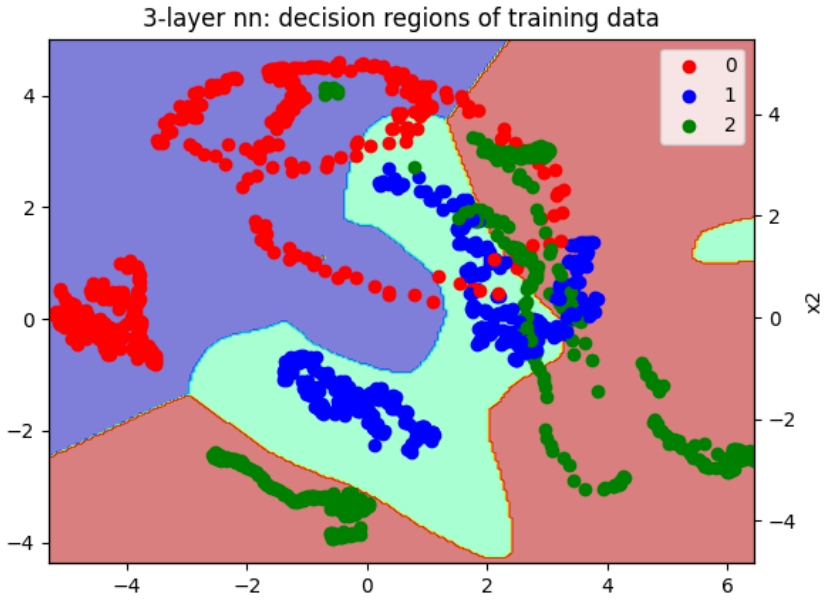
decision regions的做法是將整張圖的座標取均勻的sample，本次實作我是取250*250，再將所有座標值使用訓練好的NN做預測，並利用plt.contourf切割範圍，最後再將正確的traning data和testing data疊圖進去，即可大致看出NN的預測結果是否準確。



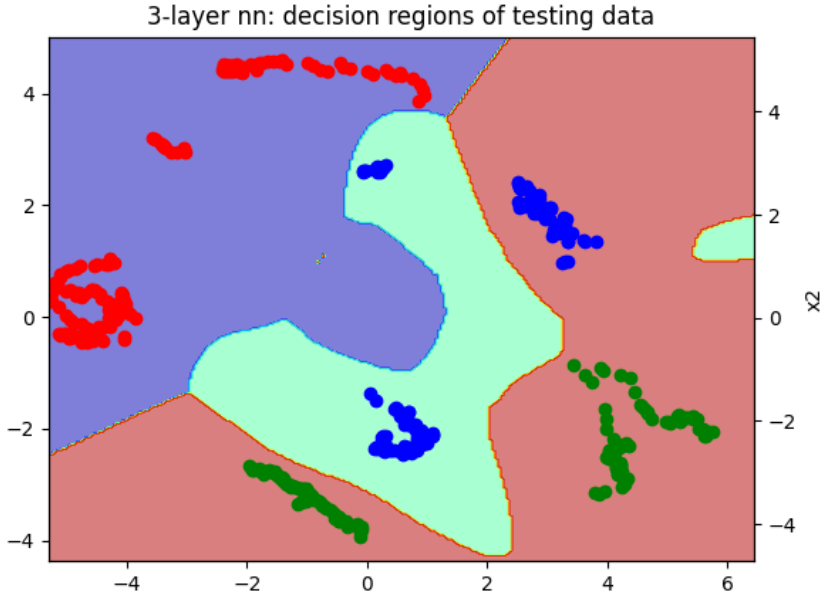
(圖4)兩層NN training data 的 decision regions



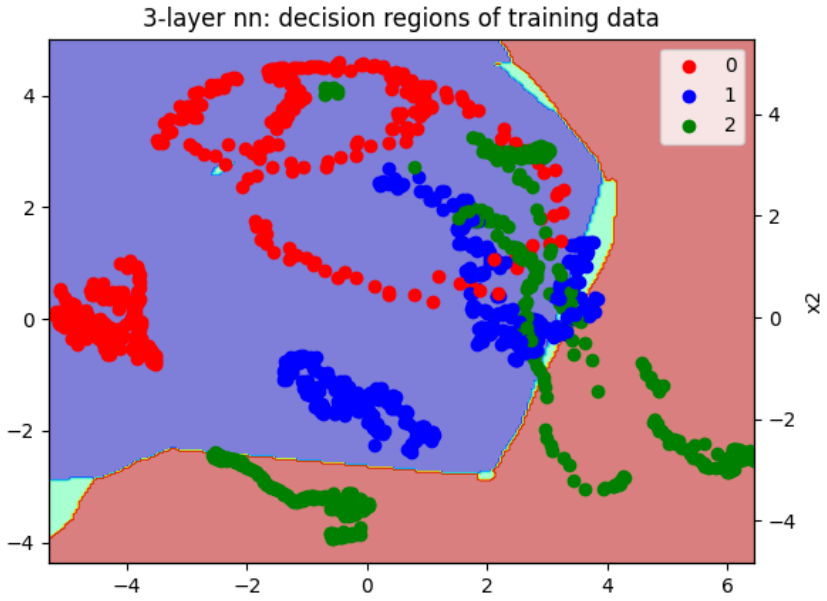
(圖5)兩層NN testing data 的 decision regions



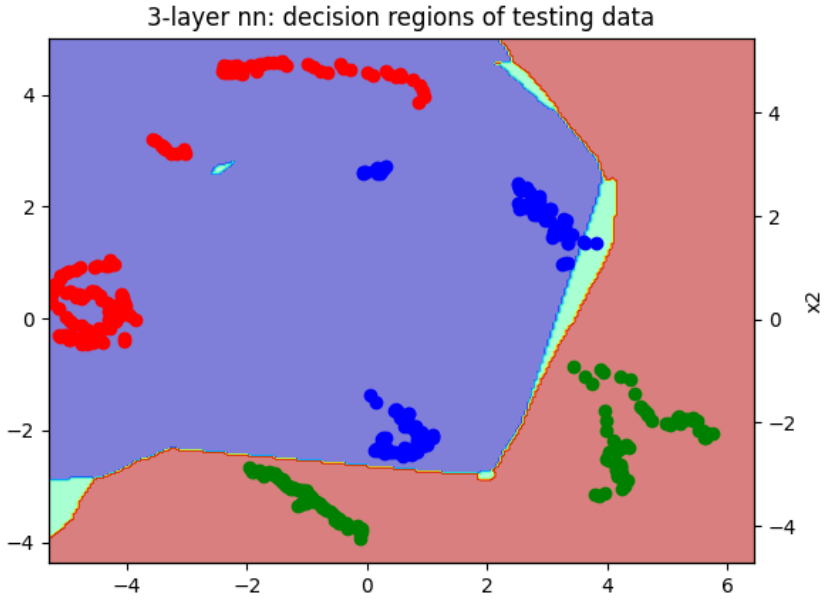
(圖6)三層NN training data 的 decision regions



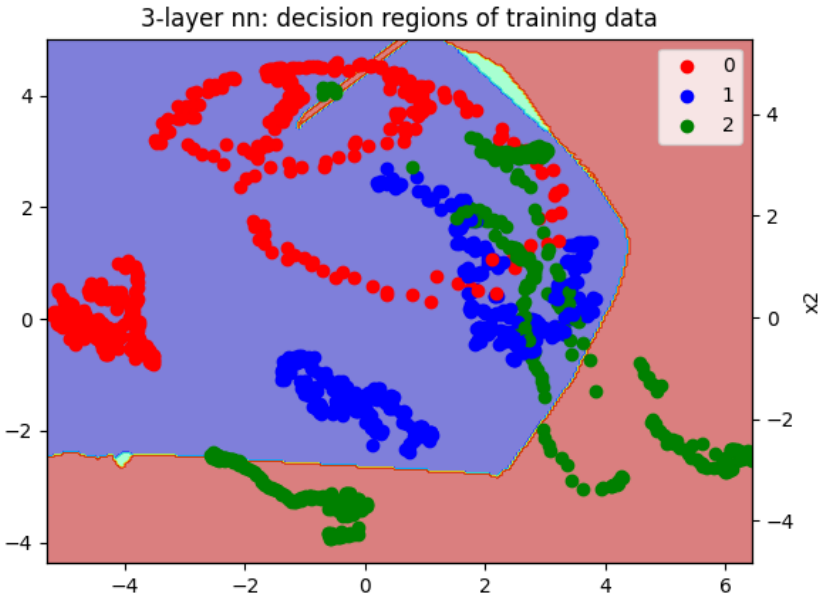
(圖7)三層NN testing data 的 decision regions



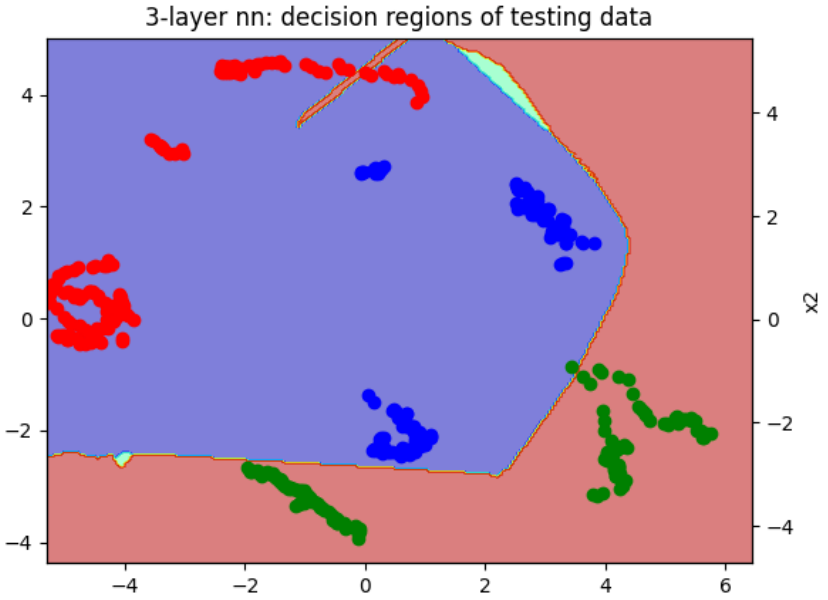
(圖8)三層NN training data 的 decision regions，使用隨機權重乘上0.9



(圖9)三層NN testing data 的 decision regions，使用隨機權重乘上0.9



(圖10)三層NN training data 的 decision regions，使用隨機權重乘上1



(圖11)三層NN testing data 的 decision regions，使用隨機權重乘上1

由以上四圖可以得知，若初始的隨機權重乘上太大的數值，就會導致模型不準確，甚至可能導致loss出現inf或0。