類神經網路 Neural Networks

作業二

學號 112522101 資工碩一

姓名 吳宥俞

目錄 Outline

1. 基本題(使用RBF網路+LMS迴歸)

1.1 GUI 功能及程式流程

GUI 部分採用 Tkinter 套件呈現，並使用 PAGE 圖型編輯器完成 UI 外觀。

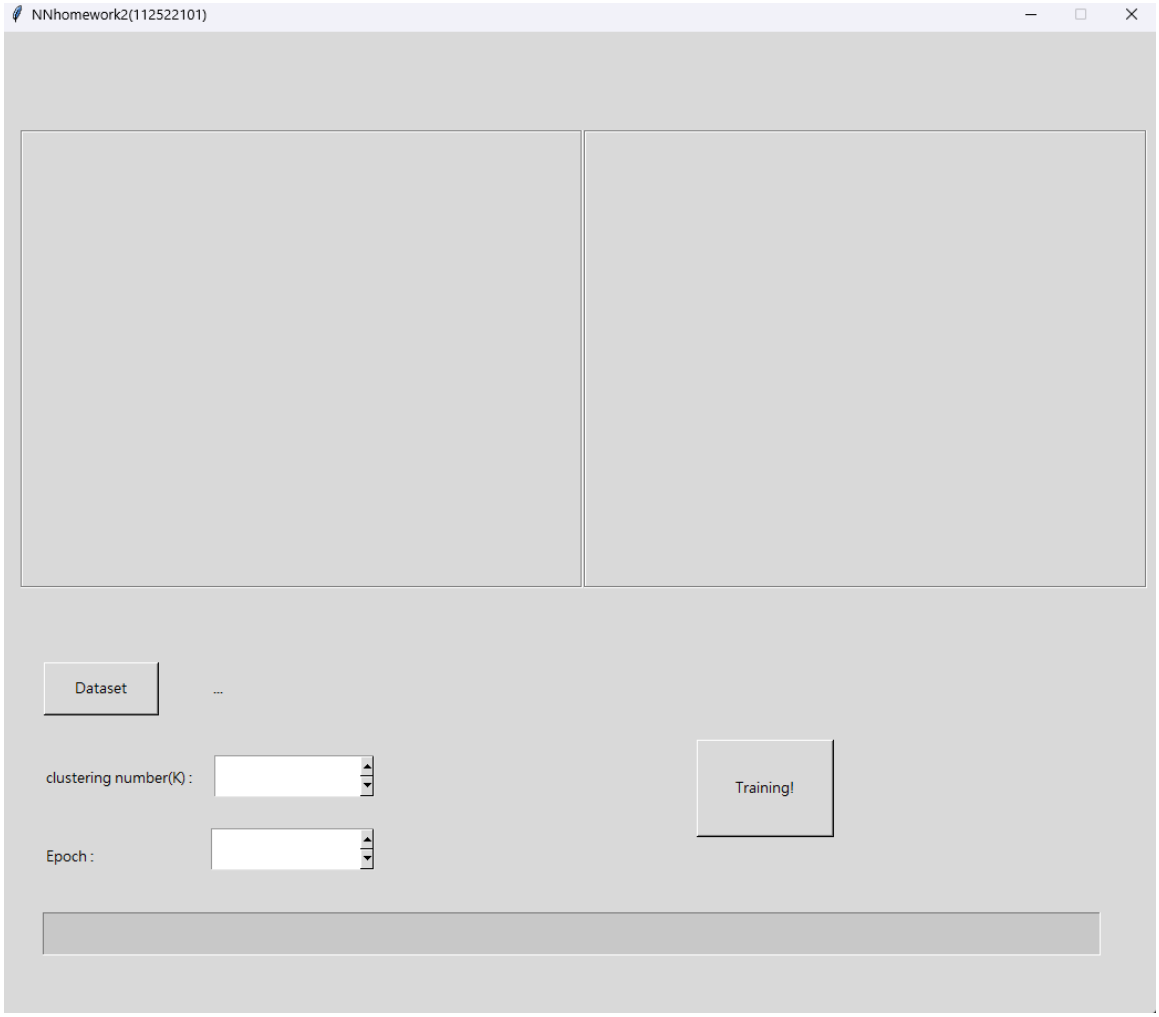


圖 1 初始外觀

使用方法:

i. 點擊 Dataset 按鈕，獲得資料集路徑。

ii. 輸入 clustering number(K)、 Epoch數值。

iii. 按下 Training!按鈕，開始訓練，下方有進度條，可得知訓練進度。

核心程式流程:

i. 當按下 Training!按鈕時，會去取得輸入的路徑、Epoch、K值，並將參數傳進 main function 進行訓練。

ii. 在訓練過程中，會將**每次訓練的結果**更新於**左邊的方框**，當中**顯示每次車子的位置、前方距離、右方距離、左方距離**。

iii. 儲存完路徑後，會將**路徑資料**的路徑軌跡結果展示於**右邊的方框**，當中**顯示車子移動的軌跡**。

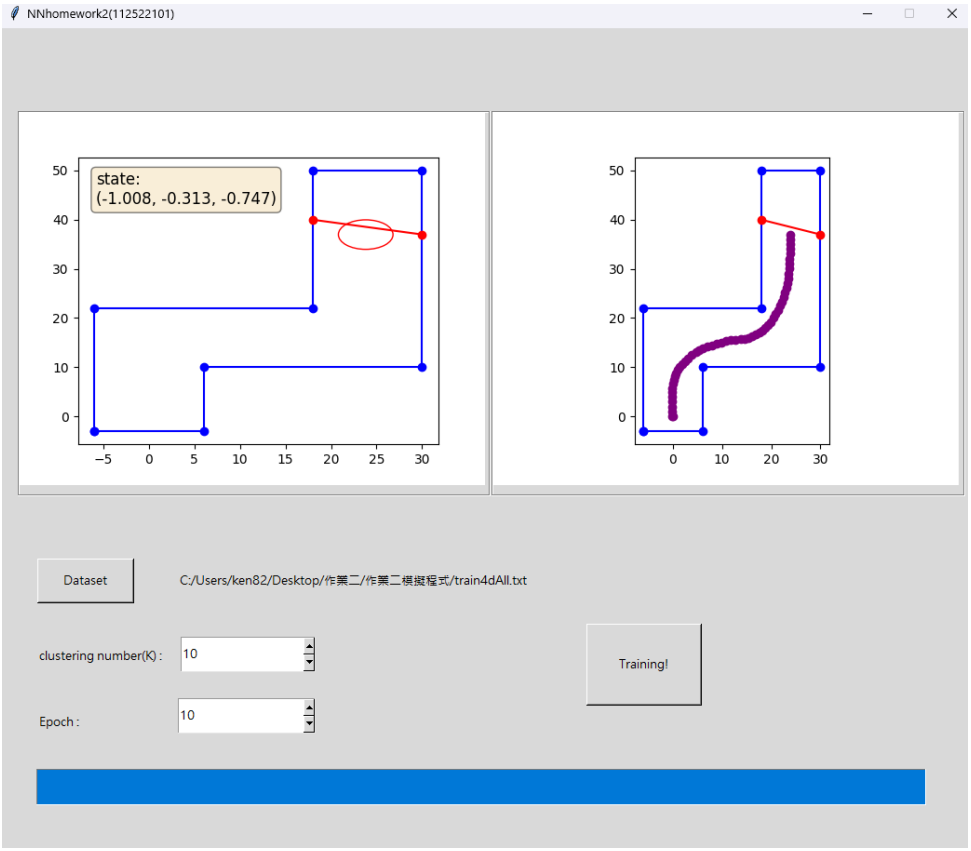


圖 2 流程完成圖(**註:左方車子會隨著每次計算結果一直更新**)

1.2 主要 function 說明

作品架構主要有 5 個檔案，5 個檔案皆相互 import:

i. UIhw2.py:UI 外觀程式碼(大部分都是 PAGE 自動生成)。

ii. UIhw2_support.py:事件反應程式碼(如按鈕點擊後會做甚麼)

iii. draw.py:作圖程式碼

iv. RBFmodel.py:類神經網路程式碼(RBF、LMS)

v. simple_playground.py:預設主程式，各種狀況判定

UIhw2.py、UIhw2_support.py 重點程式碼:

```
def trainbtn(*args):
    if _debug:
        #獲取路徑、epoch、k的數據
        path = str(_w1.l1.get())
        k = int(_w1.Spinbox1.get())
        epoch = int(_w1.Spinbox2.get())
        #呼叫程式
        simple_playground.run_example(path,k,epoch)
```

(ii.)中的 Training!按鈕，觸發後將參數傳入(v.)

RBFmodel.py 重點程式碼:

```
import numpy as np
import pandas as pd
```

使用到 pandas、numpy 套件

```
def kmeans(X, k, max_iters=100):
    # 隨機初始化中心
    centers = X[np.random.choice(range(len(X)), k, replace=False)]
    for i in range(max_iters):
        # 計算每個點到中心的距離
        distances = np.linalg.norm(X[:, np.newaxis] - centers, axis=2)
        # 分配每個點到最近的中心
        labels = np.argmin(distances, axis=1)
        # 更新中心
        new_centers = np.array([X[labels == j].mean(axis=0) for j in range(k)])
        # 如果中心不再變化，結束迭代
        if np.all(centers == new_centers):
            break
        centers = new_centers

    return centers, labels
```

利用Kmeans演算法先找出K個群中心的位置，以利後續代入RBF基底函數。

```
def rbf_weight(x, centers, gamma=1.0):
    if len(x.shape) == 1:  # 單點輸入
        return np.exp(-gamma * np.linalg.norm(x - centers, axis=1)**2)
    else:  # 多點輸入
        return np.exp(-gamma * np.linalg.norm(x[:, np.newaxis] - centers, axis=2)**2)
```

RBF基底函數:$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x} - \mathbf{x}'||_2^2)$

```
def lms_regression(X, y, learning_rate=0.01, num_epochs=1000):
    num_features = X.shape[1]
    weights = np.random.randn(num_features)

    for epoch in range(num_epochs):
        for i in range(X.shape[0]):
            prediction = np.dot(X[i], weights)
            error = y[i] - prediction
            weights += learning_rate * error * X[i]

    return weights
```

LMS回歸:鍵結值的更新按照投影片調整。

```python
def model(data,k=3,num_epochs=1000):
    #輸入資料
    df = pd.DataFrame(data)
    X_train = df.iloc[:,-4:-1].values
    y_train = df.iloc[:,-1].values

    #正規化
    X_train = (X_train - np.mean(X_train, axis=0)) / np.std(X_train, axis=0)

    # 使用 KMeans 聚類算法找到 RBF 中心
    centers,label = kmeans(X_train, k, max_iters=1000)

    # 計算 RBF 核
    X_rbf_train=[]
    for i in X_train:
        X_rbf_train.append(rbf_weight(i, centers))
    X_rbf_train = np.array(X_rbf_train)

    # 使用 LMS 算法進行回歸
    weights = lms_regression(X_rbf_train, y_train, learning_rate=0.01, num_epochs

    return weights,centers
```

整體步驟:

步驟一：輸入資料

步驟二：對訓練資料進行正規化

步驟三：利用Kmeans演算法先找出K個群聚中心

步驟四：將每個群聚中心代入RBF基底函數，投影到另一個坐標系

步驟五：將投影完的訓練資料迭代LMS回歸模型，找出誤差值最小的鍵結值

simple_playground.py 重點程式碼:

```
def run_example(trainfile,k=10,epoch=100):
    UIhw2_support._w1.Button3['state'] = tk.DISABLED
    # use example, select random actions until gameover
    p = Playground()
    #輸入資料
    df = pd.read_table(trainfile, sep=' ',header=None)
    weights,centers = RBFmodel.model(data=df,k=k,num_epochs=epoch)
    UIhw2_support._w1.progressbtn.set(40)
    df = pd.DataFrame(df)
    if df.shape[1] == 6:
        filename = "./track6D.txt"
    else:
        filename = "./track4D.txt"
    X = df.iloc[:,-4:-1].values
    state = p.reset()
    angle=[]
    posXY = [(0,0)]
    cnt=1
    coordinates,firstAndLast = draw.read_coordinates_from_file('./軌道座標點.txt')
    fig1 = plt.figure(figsize=(5, 4))
    plt.ion()
    ax1 = fig1.add_subplot(1,1,1)
    with open(filename, 'w') as f:
        while not p.done:
```

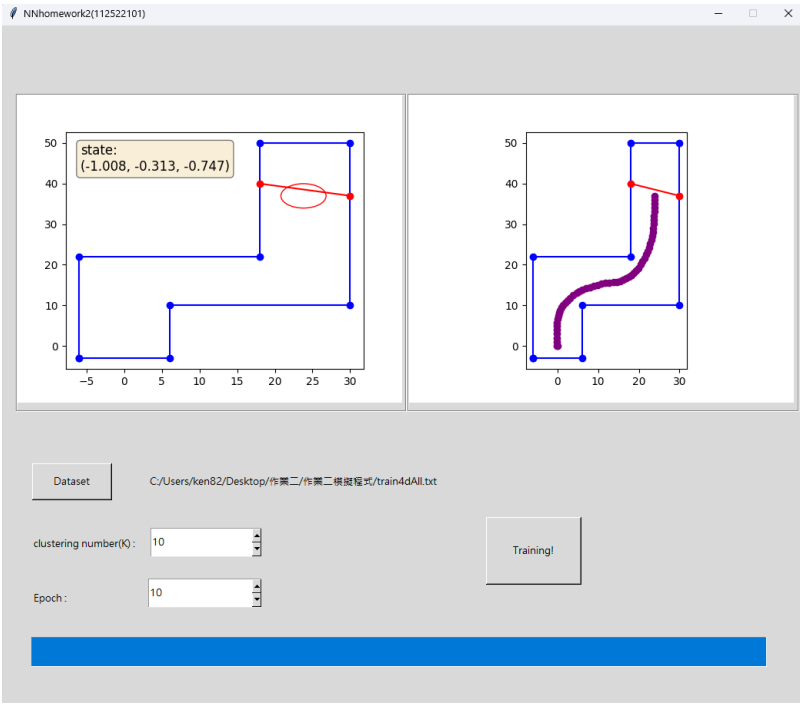
主要都寫在 run_example()的函式內，內容為呼叫RBF函式、透過matplotlib作圖、連結Tkinter畫布、輸出檔案等等。

1.3 實驗結果及分析

我發現角度輸出為40的時候，車子才是直直的往上開，所以我有將模型預測出的角度再做一些數值調整，我將(角度*4+40)當作輸出，就可以成功跑到終點了。

模型輸出的數值以下將針對每一個資料集進行分析:

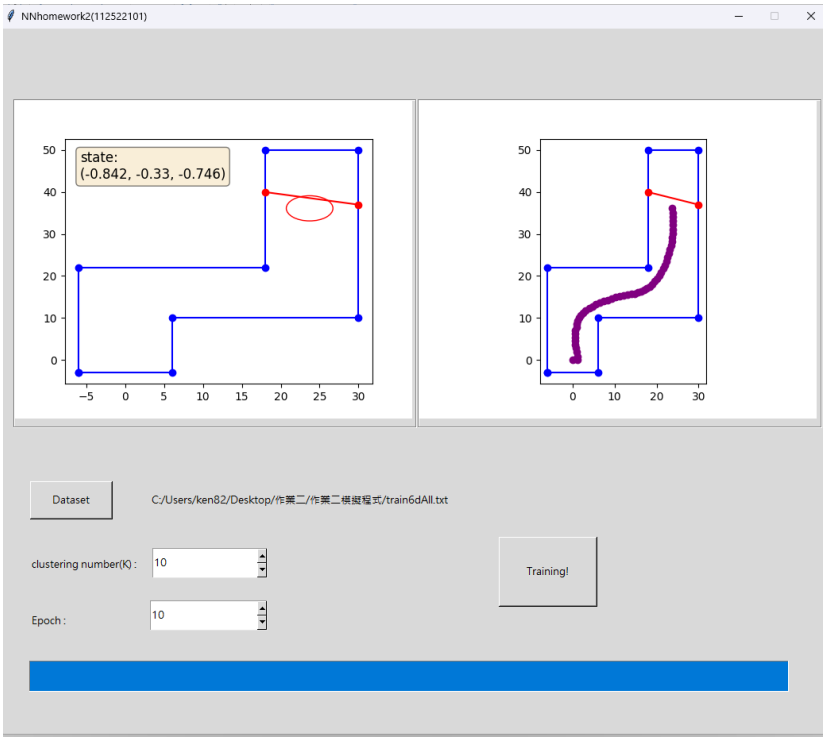
train4dAll.txt(成功跑到終點)



此資料集內容只有3個方位的距離，如果Epoch都固定10次的情況下，K值設定成大於10的數值，成功機率比較穩定，如果設定小於10個群心的話大約3次才會成功一次。

輸出的Track4d.txt檔案(3個方位的距離+輸出角度)

train6dAll.txt(成功跑到終點)



此資料集內容除了3個方位的距離，還有車子當下的座標，**但是訓練起來差別不大**。一樣如果Epoch都固定10次的情況下，K值設定成大於10的數值，成功機率比較穩定，如果K=20，基本上每次都成功。如果設定小於10個群心的話平均8次才會成功一次。

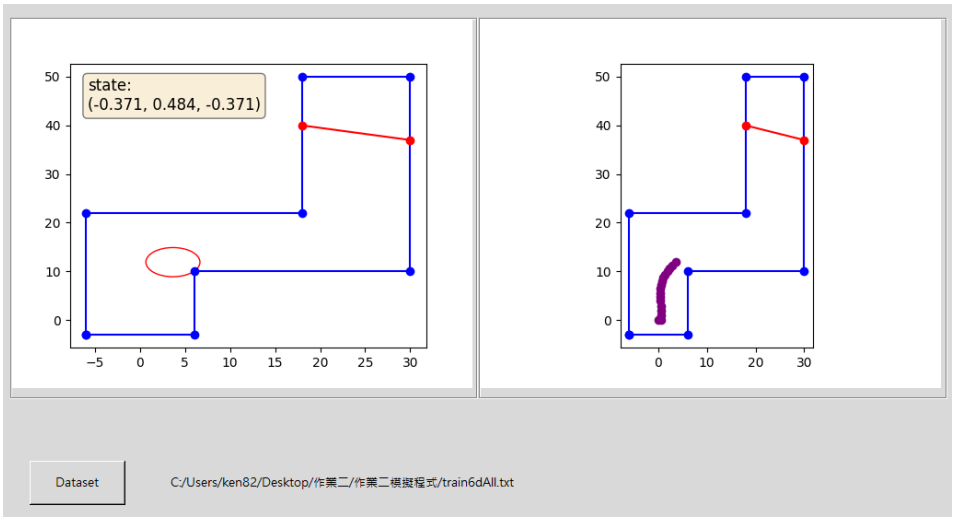
track6D.txt

檔案 編輯 檢視

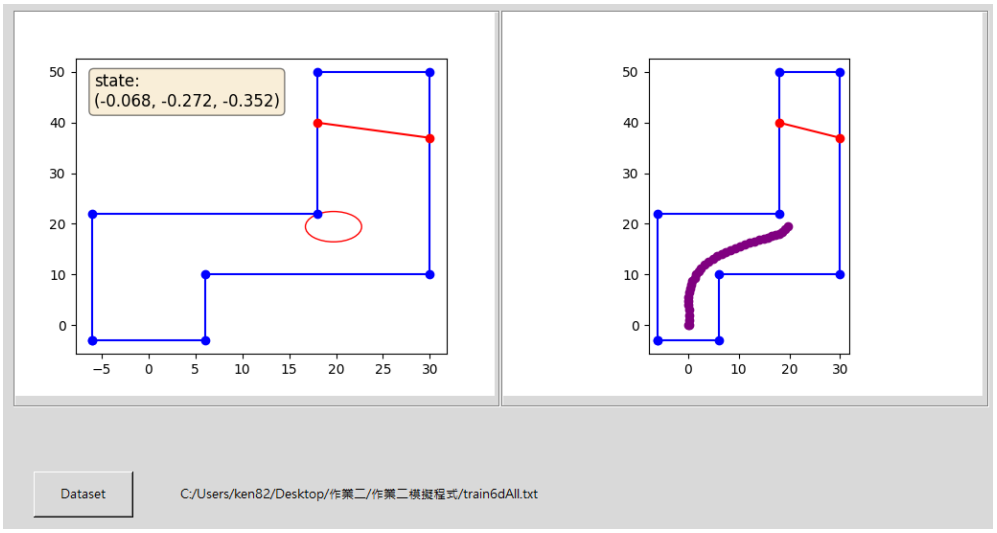
```
0.6304996582394011 0 0.794247923226278 -0.6207619141827945 -0.47720810804254066 20.937251265108156
0.6304996582394011 0.9451614555072266 0.6136350855564303 -0.5444725141214005 -0.5593044863755692 30.44022856602923
0.5589290171487549 1.9286736653856986 0.4272067669982018 -0.4843457135429232 -0.6071247245505873 38.12584813136699
0.4496498630628536 2.9221466121863804 0.2274835550146102 -0.4527129828620269 -0.6322703412529088 42.13472062922521
0.3331719522259365 3.9146411396824634 0.021109581967896765 1.3069516228792666 -0.6449986579878668 86.61865086364435
0.2501835907191076 4.676177101202257 -0.15313403946695578 3.466442307149103 -0.5058189920093533 54.02800279148855
0.28392471004887987 5.64576756756698 -0.33904185334247 3.227230074633978 -0.421547714571614 69.69094892839863
0.3608585331962704 6.511063964361045 -0.4645420505272793 2.8367386049857024 -0.19360991732157024 100.8267735540928
0.5122762786007666 7.261994557165789 -0.49600892373684574 -0.8084292167646046 0.3012815017136481 -2.253213899157792
0.7694055138499116 7.983596002018594 -0.7695618870405286 2.7761510961894356 -0.0947314383811652 112.73257488302826
0.9208232592544079 8.734526594823338 -0.8134261466150825 2.4171432359555185 0.2335129868854164 115.91177007543128
1.1779524945035527 9.456128039676143 -0.7928476586165224 2.1675979046220157 0.025365371606125297 130.24798244150077
1.53551978118951 10.133601082922521 -0.693195046066384 2.0007226000736966 -0.12414066933061908 132.64556403710233
1.986191806703279 10.753051495992674 -0.48472513950194446 1.9018892153718174 -0.22662239034997067 125.38483031335367
2.520725782427771 11.301775030780123 -0.10760715796696067 -0.32055586432227934 -0.2880064572479844 31.155409034959272
3.3042446103498335 11.903819614520891 -0.4496904816253229 0.8931429310983209 -0.3746152508098335 90.56236397273162
3.895375600694537 12.391044656044372 2.529505534434586 -0.3131364151182376 -0.408355538900524 61.7048298189589
4.689390186143365 12.873507348406697 2.073338608920528 -0.4360370579664876 -0.44466776681153486 56.20922858199067
5.548227067699247 13.303015027692688 1.7214001207662977 -0.4689832170369841 -0.47691664775105236 44.20942628359532
6.466158056459186 13.692905789375915 1.4838531562117945 -0.3875085735397036 -0.5251154861712842 44.26744267144939
7.390257320425928 14.067743610889936 1.2490226700046985 -0.31701449313006225 -0.5713404676687949 45.614579353774964
8.31853968646918 14.42651922129693 1.0087112481638065 -0.27048698480805144 -0.6128446640538825 46.66755620875864
9.25255414498378 14.764359192183395 0.7670727016494401 -0.24409170432476504 -0.6493489716733639 48.29950821672206
10.191454805753075 15.076816050805553 0.5235090746594193 -0.2415941217026682 -0.6786736353080338 50.15542963458155
11.134916761321712 15.357513610326 27 0.28058586279346115 -0.26164532898579546 -0.6986871826350946 50.93086432316385
12.086253138791522 15.600416247240057 0.04452233269404086 -0.2917148802064011 -0.7100310892624087 48.81255477919793
```

輸出的Track6d.txt檔案(車子座標+3個方位的距離+輸出角度)

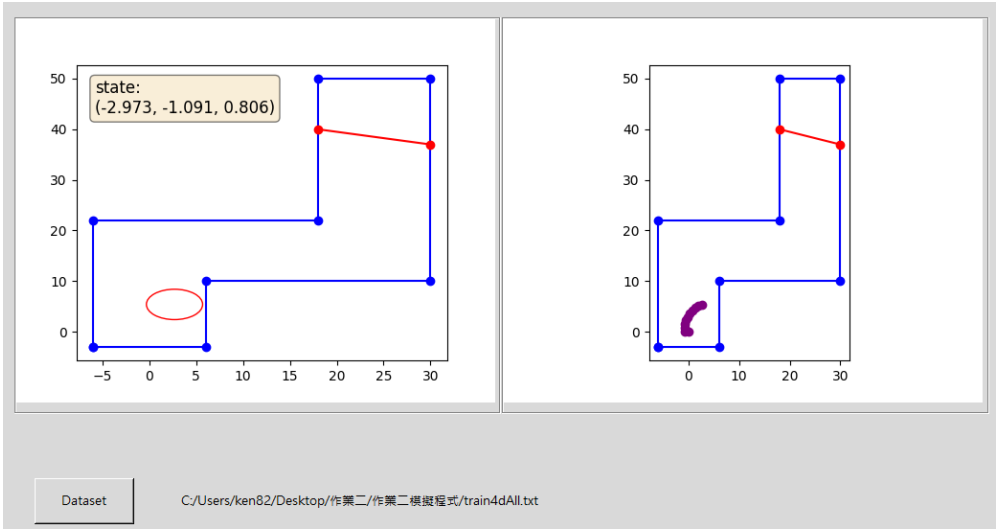
Extra:以下為兩種資料集訓練失敗的案例:



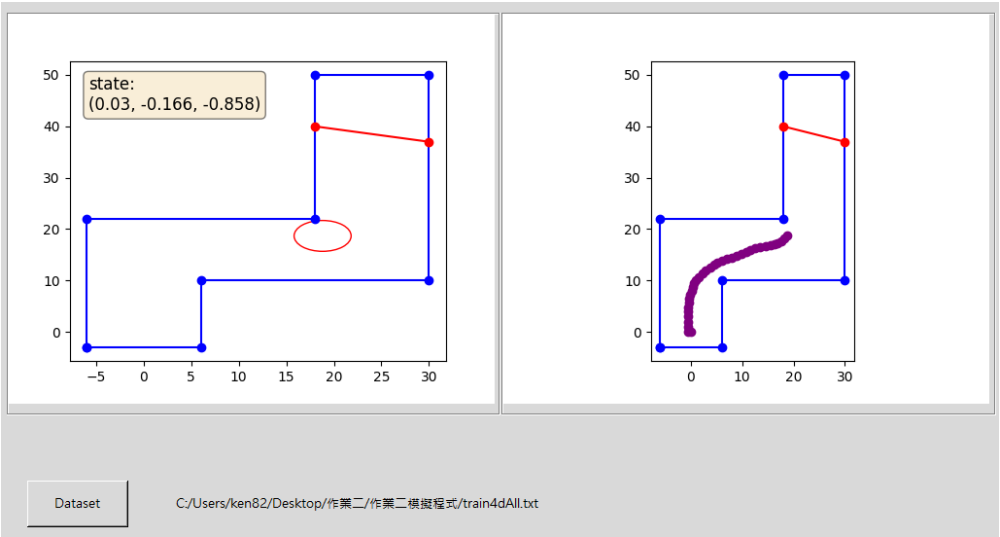
Train6dAll第一個彎道撞牆



Train6dAll第二個彎道撞牆



Train4dAll第一個彎道撞牆



Train4dAll第二個彎道撞牆

心得結論：

在這次的實作自駕車練習中，有沒有車子的座標對於整個結果沒有太大的影響，但是我認為有車子的座標在一些同寬度但卻不同位置的軌道下表現會更準確，可能是今天的軌道比較簡單，所以兩種訓練資料差別不大。

實作RBF的時候，找出適合的K值跟我預想的不太一樣，**原本以為因為輸出-40~40有80個整數點，所以高斯的局部高點要有80個可能表現才會很好，但結果是只要K值10個左右表現就很好了**，此結果令我意外，代表對於這台車子來說，方向盤只要打10個差別較大的角度就可以走到終點。

PS:原(.exe)檔案破百 MB，所以有先壓縮了，故打開需要等一陣子。