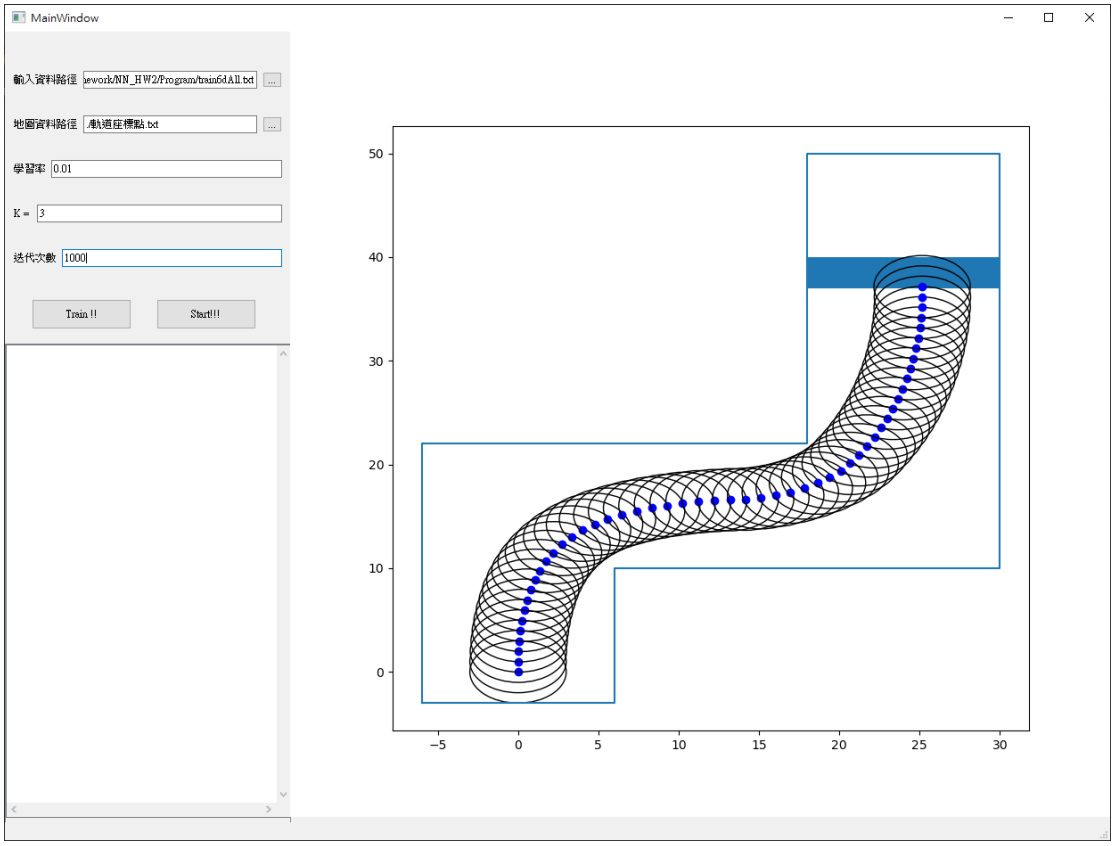
作業二：自駕車

111522094 資工碩一 涂建名

一、程式介面說明:



輸入路徑資料: 可以輸入要進入 RBFN 訓練的資料

地圖資料路徑: 可以選擇地圖的資料路徑

學習率: 設定 RBFN 的 learning rate

K: 設定將 RBFN 中的 m 初始化的 K-means 演算法的群聚數量

迭代次數: RBFN 需要跑幾個 epoch

二、程式碼簡介:

1. rbfn.py:

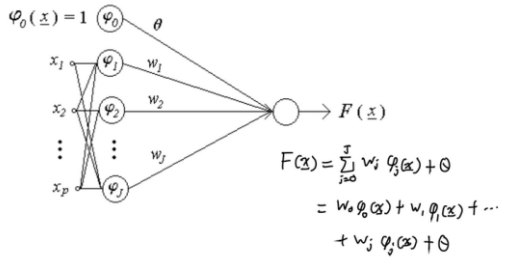
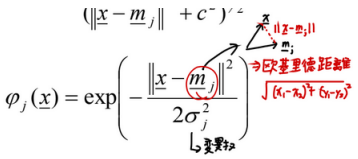


圖3.12：放射狀基底函數網路。



實作 RBFN 的 py 檔，我先用 K-means 演算法的群聚中心和群聚與群聚中心的平均距離來初始化 m 和 σ ，之後計算結合 m、 σ 、X 三個矩陣來計算出 φ 矩陣，之後計算 F = φ • W + θ 完成前傳導的過程。

利用 $E(n)=\frac{1}{2}(y_n-F(\underline{x}_n))^2$ 來當作 loss 值並實作

$$\underline{w}(n+1)=\underline{w}(n)+\eta(y_n-F(\underline{x}_n))\underline{\varphi}(\underline{x}_n)$$

$$\underline{m}_j(n+1)=\underline{m}_j(n)+\eta(y_n-F(\underline{x}_n))w_j(n)\varphi_j(\underline{x}_n)\frac{1}{\sigma_j^2}(\underline{x}_n-\underline{m}_j(n))$$

$$\sigma_j(n+1)=\sigma_j(n)+\eta(y_n-F(\underline{x}_n))w_j(n)\varphi_j(n)\frac{1}{\sigma_j^3}\left\|\underline{x}_n-\underline{m}_j(n)\right\|^2$$

$$\underline{\varphi}(\underline{x}_n)=[\varphi_0(\underline{x}_n),\varphi_1(\underline{x}_n),\cdots,\varphi_J(\underline{x}_n)]^T$$

$$\varphi_0(\underline{x}_n)=1$$

$$\underline{w}(n)=[\theta(n),w_1(n),\cdots,w_J(n)]^T$$

來完成倒傳遞的調整。

2. drawplot.py

畫出 GUI、模擬車輛移動的程式，

Sensor 的作法:

事先設定一個固定的 vector = [100, 0]

在利用旋轉矩陣，計算長度為 100、角度為Φ(t)的 rotated_vector

之後計算所有邊界與 a[rotated_vector] + [車輛位置]的所有交點，選出與車輛位置最近的交點的歐基里德距離作為該方向 sensor 偵測出來的距離

車輛的移動方程式:

$$x(t+1) = x(t) + \cos[\phi(t) + \theta(t)] + \sin[\theta(t)]\sin[\phi(t)] \quad (10.18)$$

$$y(t+1) = y(t) + \sin[\phi(t) + \theta(t)] - \sin[\theta(t)]\cos[\phi(t)] \quad (10.19)$$

$$\phi(t+1) = \phi(t) - \arcsin\left\lfloor \frac{2\sin[\theta(t)]}{b} \right\rfloor \quad (10.20)$$

其中 Φ(t) 是模型車與水平軸的角度， b 是模型車的長度，x 與 y 是模型車的座標位置， θ (t) 是模型車方向盤所打的角度

碰撞偵測:
由三個 Sensor 回傳回來的距離判斷是否小於車身的半徑 3

3. main.py
   UI 的主程式
4. toolkit.py
   放有歐基里德距離計算、旋轉矩陣計算、向量和直線交點計算的程式

三、實驗結果

1. 實驗一:

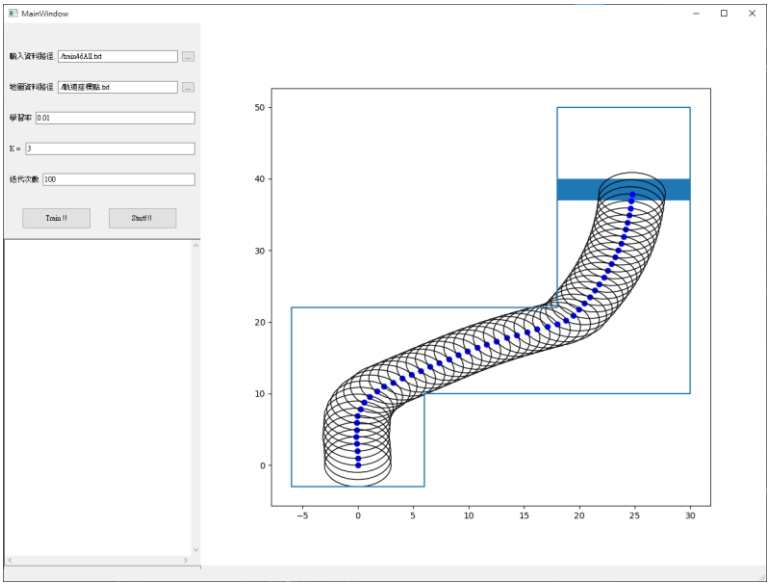
使用資料集: train4dAll.txt

Learning rate: 0.01

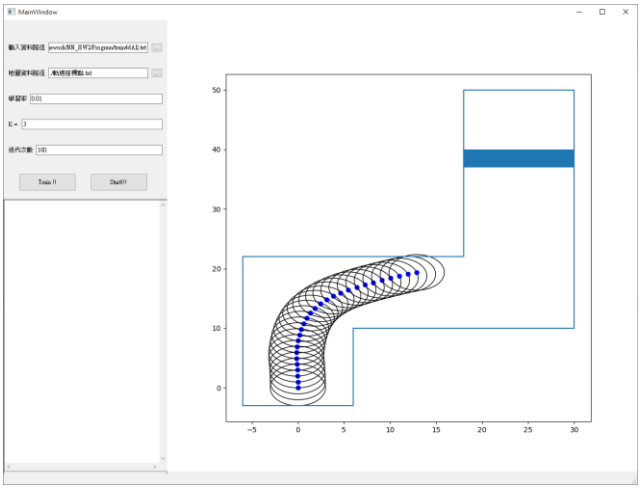
K = 3

迭代次數 = 100

優點:訓練速度快



缺點: 自走車碰壁機率較高(大概:19/20)



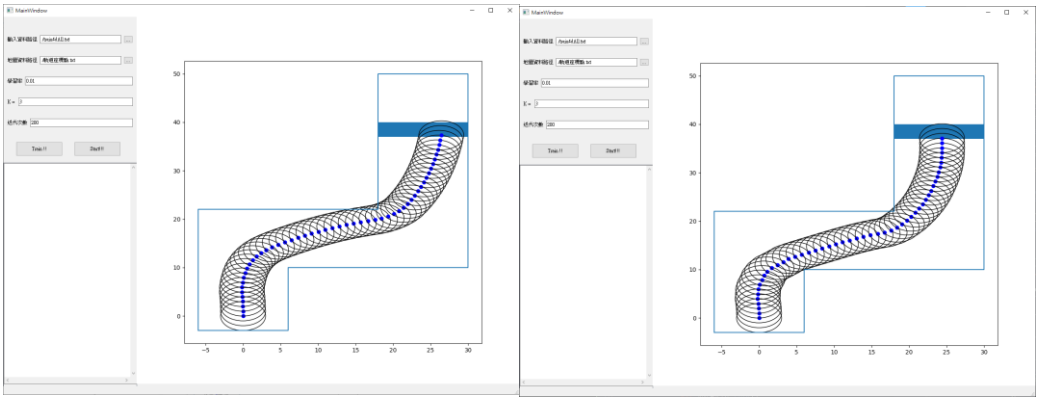
2. 實驗二:

使用資料集: train4dAll.txt

Learning rate: 0.01

K = 3

迭代次數 = 200



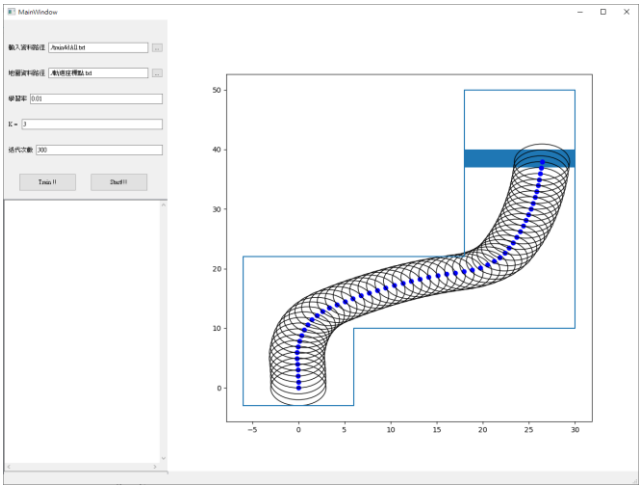
3. 實驗三

使用資料集: train4dAll.txt

Learning rate: 0.01

K = 3

迭代次數 = 300



優點: 成功機率高且軌跡較實驗一漂亮

缺點: 需要時間較實驗一長

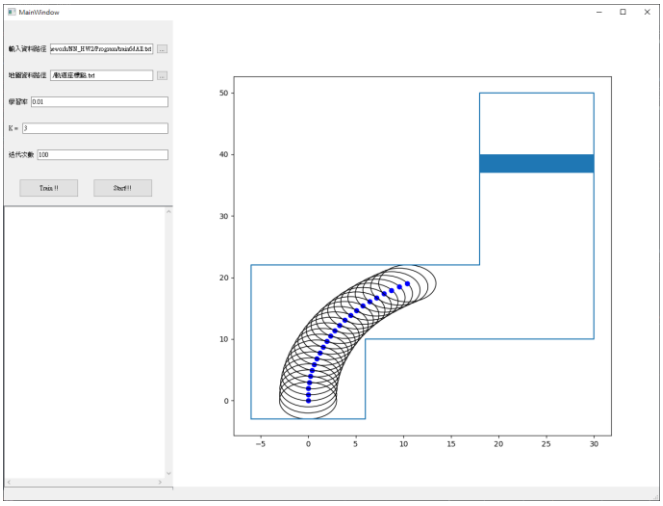
4. 實驗四:

使用資料集: train6dAll.txt

Learning rate: 0.01

K = 3

迭代次數 = 100



結果: 失敗機率高

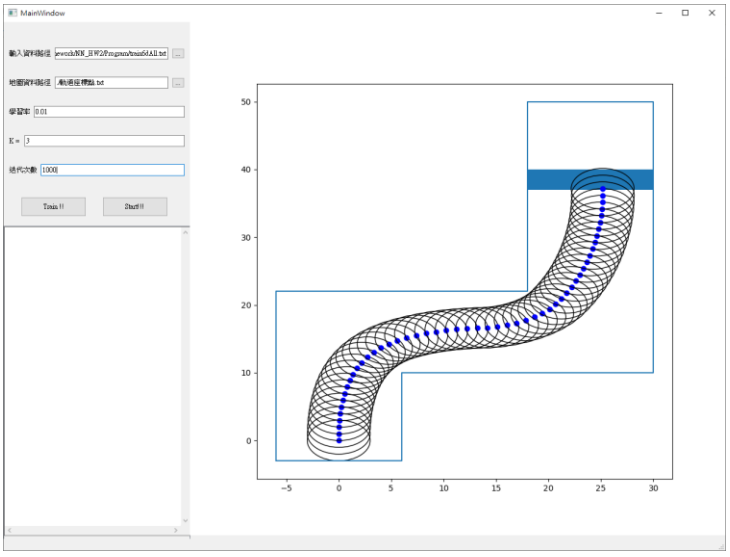
5. 實驗五:

使用資料集: train6dAll.txt

Learning rate: 0.01

K = 3

迭代次數 = 1000



優點: 得出的結果最好、連牆角都不會碰撞到

缺點: 需要時間過長(大概 1 分鐘至 3 分鐘之間)

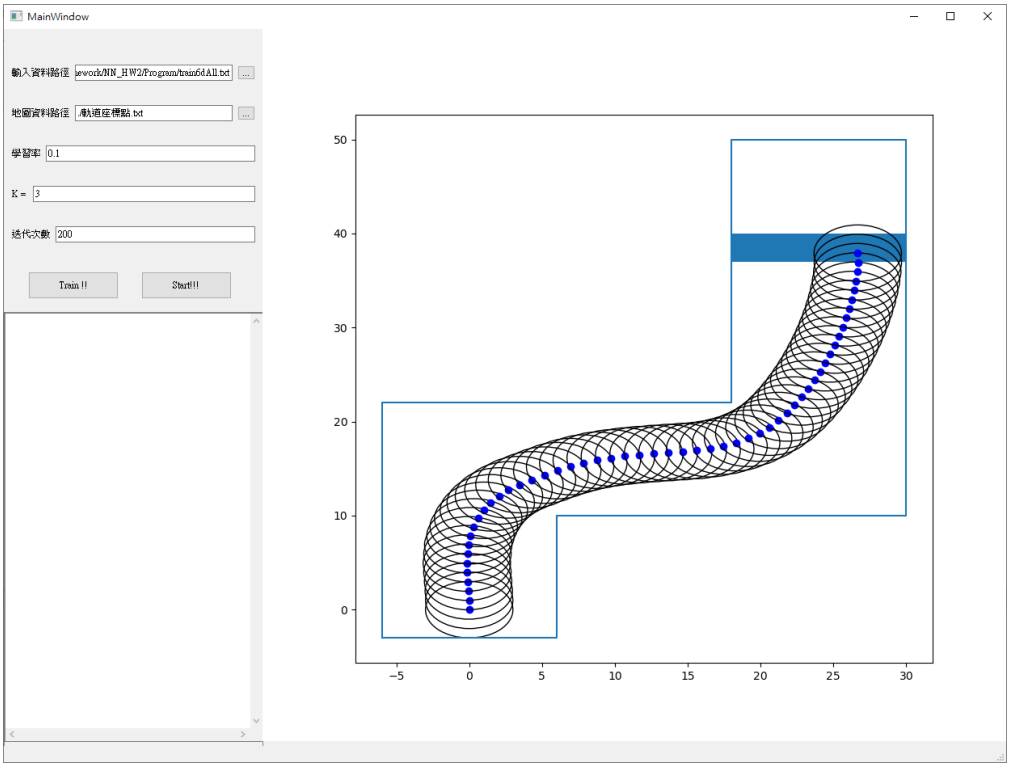
6. 實驗六:

使用資料集: train6dAll.txt

Learning rate: 0.1

K = 3

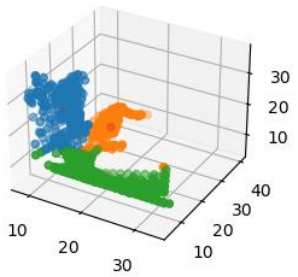
迭代次數 = 200



結果: 所需時間短、軌跡完美、沒有撞到任何牆

分析:
從實驗來看 RBFN 的 loss 會隨著 learning rate 的降低而下降，目前找到適合的 learning rate 為 0.1 or 0.01，對比於 MLP RBFN 的速度較快，並且也能跑出非線性的預測。
如果單論這次自走車最好的參數為
使用資料集: train6dAll.txt
Learning rate: 0.1
K = 3
迭代次數 = 200
train6dAll.txt 由於比 train4dAll.txt 多了 x, y 的位置資料所以預測出的軌跡相較於 train4dAll.txt 也會比較漂亮

而 learning rate 目前觀察 0.1 會下降快速，但是如果調到 0.3 的時候 loss 就會暴增，調到 0.01 時則下降速度較為緩慢
K = 3 從資料的分布來看分三群顯然是比較合理的選擇



迭代次數的話目前觀察則是越大越好，但是相對付出的時間成本就會更高。

如果使用的訓練資料集為 train4dAll.txt
目前檢測最好、最有效率的結果為
Learning rate = 0.01
K=3
Epochs = 200~300