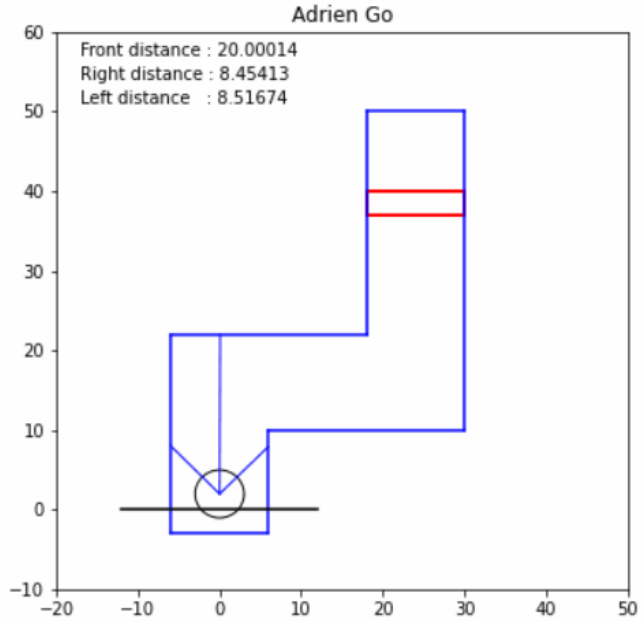
Neural Network HW2 Report

資工碩一 郭佩昇
110522071

1 程式介面說明

打開檔案”Adrien_Go_(0,0).gif”，會跑出一張 gif 動畫，如圖片 1.



圖片 1.| 程式 UI 介面

- 藍色線段是軌道邊界，黑色線是起始線，紅色框框是終點範圍，黑色圓形代表車子。
- 圖片的左上角顯示分別顯示前、右、左三個方向的距離。
- 開始移動後，車子會有三條藍線延伸出去，分別是前、右、左三個方向的指向線。

註: Adrien_go_(0,0).gif 代表車子從(0,0)開始走
Adrien_go_(2,0).gif 代表車子從(2,0)開始走
Adrien_go_(-2,0).gif 代表車子從(-2,0)開始走

2 程式碼說明

分成計算車子到牆壁的距離以及 RBF 模型，這兩個部分討論。

2.1 計算車子到牆壁的距離

- **def vehicle_move(x, y, theta, empty)**

目的：這裡描述模擬車的運動方程式
x: 車子所在的 x 座標
y: 車子所在的 y 座標
theta: 車子要轉的角度(θ)
empty: 車子與水平線的夾角(ϕ)

- **def distance_cal(vehicle_x, vehicle_y, empty)**

目的：計算車子的三個方向到牆壁的距離
vehicle_x: 車子所在的 x 座標
vehicle_y: 車子所在的 y 座標
empty: 車子與水平線的夾角

這裡會計算車子和前、右、左三個方向的距離
計算方式分成五個步驟

1. **def find_vehicle_front_all_wall_point(line)**

車子這三個方向的直線可以算出一個直線方程式，將方程式和所有牆壁的方程式會算出很多交點。

2. **def find_point_on_wall(points)**

但這些交點並不是真的都在那些牆壁上(有些是在那些牆壁的延長線上)，因此這個步驟要去判斷這些點是不是都在真的牆壁上。

3. **def find_correct_direction(points, vehicle_x, vehicle_y, empty)**

接著要分析車子現在方向是朝向上面還是朝向下面，以及是朝向正右方(水平夾角=0)，或是朝向正左方(水平夾角=180)。

4. **def find_closest_point(points, vehicle_x, vehicle_y)**

這個方向仍然可能有多個交點，因此要找離車子最近的那一個，這個交點即是正確的那個交點。

5. **def cal_dist(point, vehicle_x, vehicle_y)**

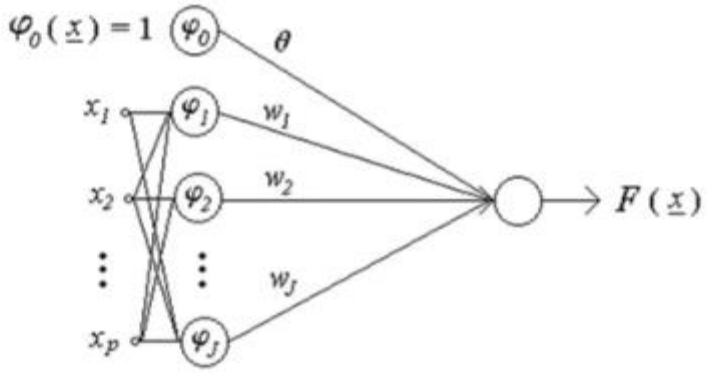
將這個交點和車子的中心去算距離，回傳算出來的距離以及交點。

2.2 **RBF 模型**

$$F(\underline{x}) = \sum_{j=1}^{J} w_j \varphi_j(\underline{x}) + \theta$$

$$\varphi_j(\underline{x}) = \exp\left(-\frac{\|\underline{x} - \underline{m}_j\|^2}{2\sigma_j^2} \right)$$

示意圖：



輸出的 F(x)是角度(θ)，因此輸出層只會有一個神經元
如果是要做分類的話，那輸出層就會有多個神經元(跟要分類的數量一樣多)

分成 K-means 和 RBFNN 來討論

- **def kmeans(X, k, max_iters)**

目的：將資料分成 k 群，算出每一群的中心點與標準差。
X: 所有的資料點
k: 分成 k 群
max_iters: 最大迭代次數

- **class RBFNN**

RBFNN.train(lr, n_epochs)
目的：訓練 RBF 模型
lr: 0.0001
n_epochs: 500

基底函數 $\varphi_j(\underline{x}) = \exp\left(-\frac{\|\underline{x} - \underline{m}_j\|^2}{2\sigma_j^2} \right)$

使用 LMS 演算法調整權重

$$E(n)=\frac{1}{2}(y_n-F(\underline{x}_n))^2$$

$$w_j(n+1)=w_j(n)-\eta\frac{\partial E}{\partial w_j(n)}=w_j(n)+\eta(y_n-F(\underline{x}_n))\varphi_j(\underline{x}_n)$$

$$\theta(n+1)=\theta(n)-\eta\frac{\partial E(n)}{\partial \theta(n)}=\theta(n)+\eta(y_n-F(\underline{x}_n))$$

RBFNN.predict(X_pred)

目的: 輸入要預測的資料，並輸出預測的角度

X_pred: 被預測的資料(前、右、左三個方向的距離)

output: 預測的角度(θ)

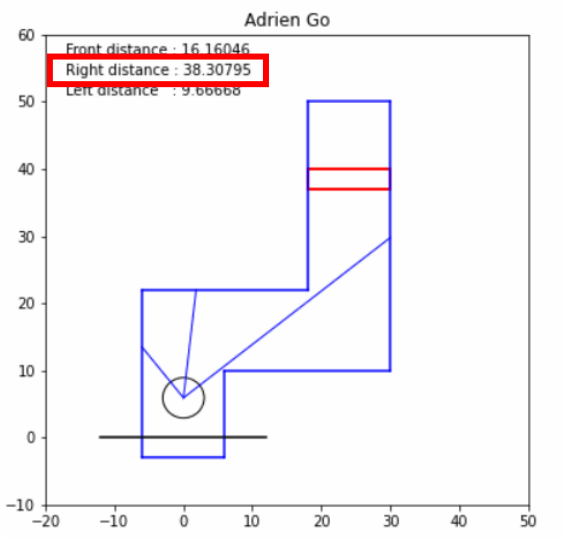
使用上方訓練完的 RBF 模型的參數，來預測我們輸入的資料。

3 實驗結果

自動車可以順利地到達終點，且不論是在起跑線上的哪一點皆可。

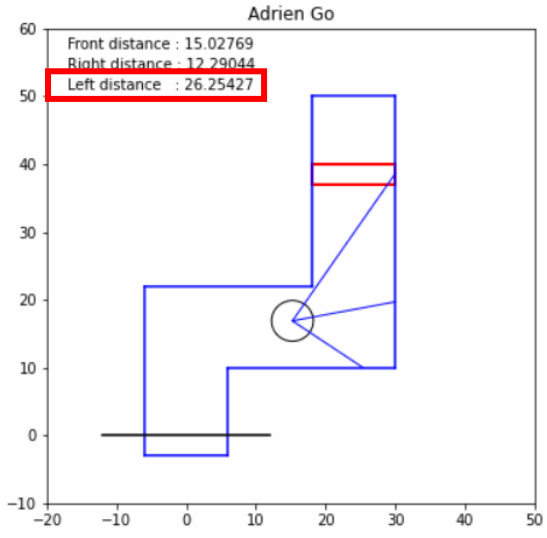
- **右轉**

當偵測到右邊的距離突然變大，則車子會開始進行右轉彎。



- **左轉**

當偵測到左邊的距離突然變大，則車子會開始進行左轉彎。



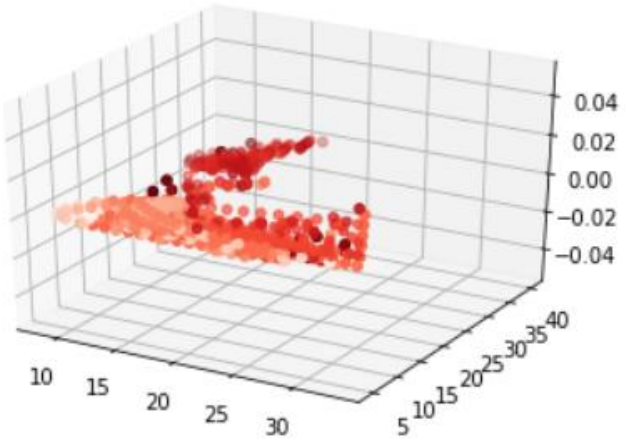
當車子偵測到有某一邊的數值突然變大時，則會開始朝那個方向進行轉彎。全部觀察下來，車子會朝距離大的那個方向進行轉彎，而距離相對小的時候會進行比較小幅度的轉彎，距離大的時候會進行相對較大幅度的轉彎。

4 分析

分成 K-means, RBF Train, RBF Predict 三個部分來討論

4.1 K-means

一開始要決定分幾群的時候，本來想說可以看圖片來分析，於是我產生下面這張圖



圖片 2.| 三個軸代表三個方向的距離。圓點的顏色的深淺代表角度的值，顏色越深代表角度越大

但是這樣子幾乎是看不出來到底分幾群比較好，於是我後來用 try & error 的方式，最後將他們分成 100 群。

4.2 RBF Train

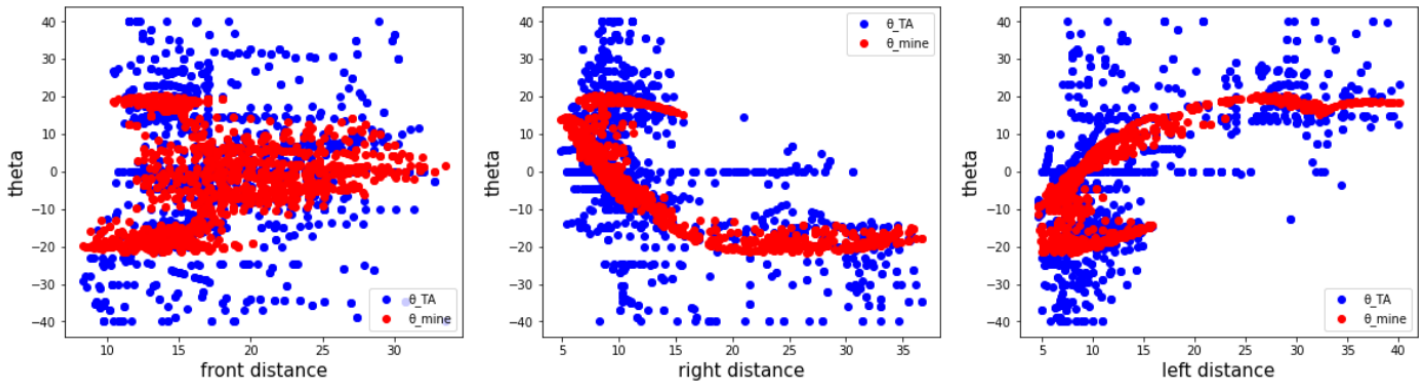
我做了兩種迭代次數，一種是 500 次、一種是 1500 次，分別探討他們的 Loss，效果如下方表格。
(Loss function 採用均方誤差-MSE)

Iteration	Loss
500	150
1500	148.23

可以看到，迭代次數高的，loss 有比較小，但是差距並沒有很大；而代入資料並且進行預測後，從自走車的路徑可以看出幾乎是沒有差異，因此迭代次數並不用設定太高，只需要讓 loss 低於一個程度就可以了，不然反而會花了很多時間，但卻沒有達到該有的效益，因此我將迭代次數設定為 500。

4.3 RBF Predict

因為沒辦法一次全部的資料都放在一張圖裡面分析，我分成下面三張圖片



圖片 3. | x 軸分別是前面方向的距離、右邊方向的距離、左邊方向的距離；y 軸都是角度(θ)；藍色點是助教提供的資料；紅色點是我的 RBF 產生的預測資料

由圖片可以看出，不論是哪一個方向，相對於助教的資料，我的資料都**沒有辦法產生大角度的輸出** (最多大概就到 25 度)，也就代表訓練出來的自動車沒有辦法做大角度的轉彎 (實測出來也是如此)。
會造成這樣的原因，我的猜想是因為大角度的資料比較少，這樣導致在第一步做 K-means 的時候，和大角度相關的資料中心點個數就會比較少，也因此後續訓練的時候，很容易就被平均掉了，所以如果要避免產生這種現象，應該要**多增加資料量**。

不過即使沒有大角度的資料，我的自動車一樣可以順利地到達終點，而它就會比較早開始轉彎，轉彎的幅度也會比較小。