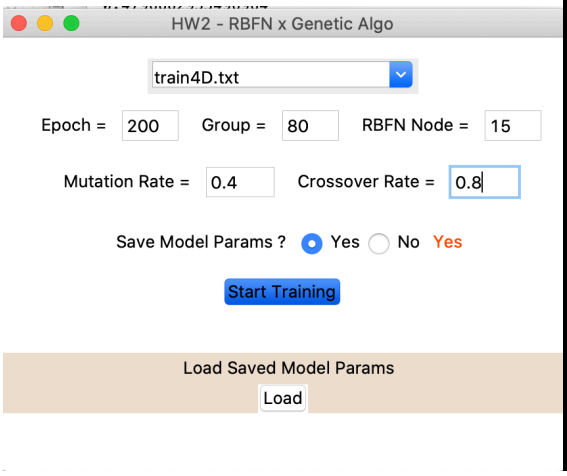
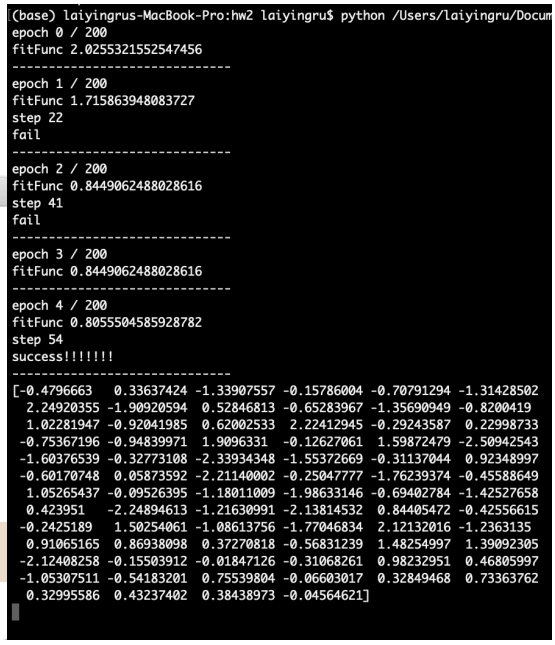
108522050 賴映如 計算型智慧作業二 RBFN&Genetic Algo 自走車

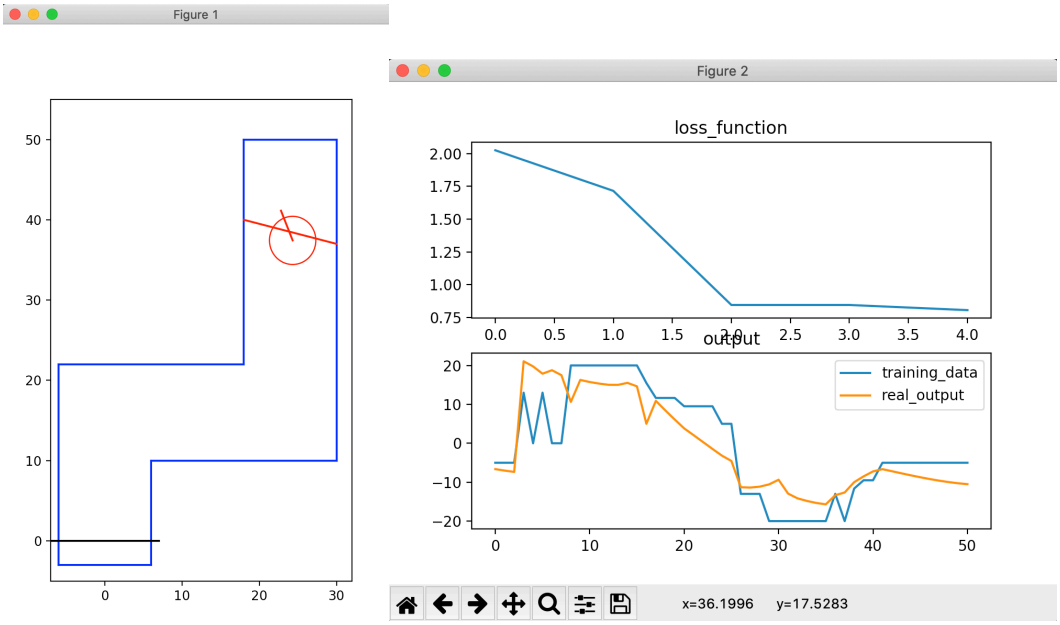
A. 程式介面/執行視窗/參數檔案說明:

1. **設定功能：**
   i. 選擇檔案，使用自己的資料集 train4D.txt / train6D.txt 附件資料集 train4dAll.txt / train6dAll.tx
   ii. 設定計算次數 Epoch、族群數 Group、RBFN Node 神經元數、交配機率（Mutation Rate）、突變機率 （Crossover Rate）
   iii. 選擇是否儲存本次訓練好的網路參數 （選後會有紅字顯示）
   iv. 按下 Strat Training 按鈕 開始訓練
2. **Command 視窗**（右圖）：將顯示訓練回數 epoch 及適應函數值 fitFunc，若當次適應函數有變化則會顯示當次進入 track 行走的步數 step





3. **結束訓練：**
   i. 達到使用者輸入 epoch 次數
   ii. 跳出當次訓練之 loss function 及 output 圖表
   iii. 若有成功訓練出可成功走出軌道的模型，彈出車子行走視窗



4. **Save/Load Model Params:**
   i. 前次訓練 save model params 選擇 Yes
   ii. RBFN_params.txt 將依照要求格式存取如上圖

Θ

第一顆神經元參數: $W_1\ m_{11}\ m_{12}\ m_{13}\ \sigma_1$

第二顆神經元參數: $W_2\ m_{21}\ m_{22}\ m_{23}\ \sigma_2$

…

第 J 顆神經元參數: $W_j\ m_{j1}\ m_{j2}\ m_{j3}\ \sigma_j$

RBFN_params.txt

```
-0.4796662955496964
0.3363742396380549 -0.29243586624504136 0.22998733407372124 -0.7536719586396062 -0.1550391156619959
-1.3390755660759046 -0.9483997074867181 1.9096331016498995 -0.12627060532896728 -0.0184712569761913
-0.1578600412434174 1.5987247905503739 -2.50942543163093 -1.6037653891629269 -0.31068260583016377
-0.7079129440941079 -0.3277310800217315 -2.3393434799410713 -1.5537266872845095 0.9823295080743136
-1.314285015402579 -0.3113704407521433 0.9234899691853903 -0.6017074770457347 0.4680599687359844
2.2492035486377664 0.058735924901703616 -2.2114000188526175 -0.2504777721379472 -1.0530751131576224
-1.9092059394584446 -1.762393744395617 -0.45588649467135917 1.0526543739167307 -0.5418320137064891
0.5284681321284188 -0.09526395448931653 -1.180110090716885 -1.9863314643259946 0.7553980360776428
-0.6528396711367989 -0.6940278444983071 -1.4252765758317467 0.42395100440061584 -0.06603016643406288
-1.3569094862136357 -2.248946129667295 -1.216309913043367 -2.138145317979304 0.3284946778239517
-0.8200418964020437 0.844054720800195 -0.42556614987863084 -0.24251889631223522 0.7336376155020526
1.0228194659586485 1.5025406124106924 -1.0861375565136422 -1.770468344297873 0.3299558559915503
-0.9204198518178122 2.1213201584935373 -1.2363134975732142 0.9106516542497294 0.4323740230555758
0.6200253287281579 0.8693809793638207 0.37270817874229084 -0.5683123860021898 0.38438972739802313
2.2241294511402647 1.4825499745088502 1.390923046997691 -2.1240825792463345 -0.045646208638712715
```

   iii. Load model params 按下 Load 按鈕，便會將 RBFN_params.txt 的網路參數丟進車子軌道，跳出視窗顯示行走路線

B. 基因演算法實作說明：

1. 使用**實數型基因演算法**
2. **輪盤式選擇複製**：依每個物種(基因向量)的適應函數值的大小來分割輪盤上的位置，適應函數值越大則在輪盤上佔有的面積也越大，每個物種在輪盤上所佔有的面積比例也就代表其被挑選至交配池的機率。
3. 交配池總數為原先設定的族群數，若複製的數量大於此，則依權重比最低的基因開始做刪減，但若複製的數量不足，則將權重比例最高的基因，依不足的數量增加基因進交配池。
4. **實數型的交配機制：** 交配的過程會使得兩個物種之間的距離變得更近或更遠。依據使用者輸入之交配率決定交配數量，若數量不為偶數則加一。

- **實數型的交配過程：**交配的過程會使得兩個物種之間的距離變得更近或更遠。

$$\begin{cases} x_1' = x_1 + \sigma(x_1 - x_2) \\ x_2' = x_2 - \sigma(x_1 - x_2) \end{cases} \quad \begin{cases} x_1' = x_1 + \sigma(x_2 - x_1) \\ x_2' = x_2 - \sigma(x_2 - x_1) \end{cases}$$

σ 是隨機選取的微量正實數。

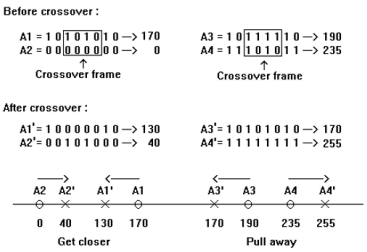


圖10.12：交配過程所造成物種移轉的情形。

5. **實數型的突變過程：** 突變過程為隨機選取一個物種，並且加入微量雜訊。 依據使用者輸入之突變率決定突變數量。

$$x = x + s \times random\_noise$$

其中 s 控制所加入雜訊之大小。

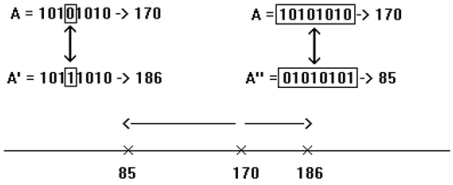


圖10.13：突變過程所造成物種移轉的情形。

C. 程式碼說明:

程式分二個檔案，分別為

- ✓ **hw2_rbfn_ga.py:** 主要 RBFN 及基因演算法運算、視窗介面。
- ✓ **carMoving.py:** 畫出車子軌道、更新車座標、角度、判斷牆壁、距離前左右方牆壁距離。

1. 處理資料與初始基因向量

i. **loadData( )** 將資料依規定正規化。

```
def loadData(fileName):
    data = np.loadtxt(fileName)
    dim = len(data[0])
    for i in range(len(data)):
        for n in range (dim - 1):
            data[i][n] = ( data[i][n] / 80 ) * 2 - 1    # 四維資料前3項:距離 值域0~80 正規化到-1~1
        data[i][-1] = ( data[i][-1] / 40 )  # 最後一項:角度 值域-40~40 正規化到-1~1

    return data
```

ii. **initWeight( )** 生成依規定格式 group 個基因向量。

Dim(j) = 1 +j + i*j +j

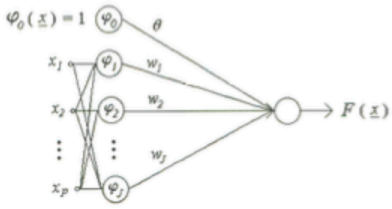
即 (θ,w_1 ,w_2 , ⋯, w_j , m_{11}, m_{12}, ⋯, m_{1i}, m_{21}, m_{22}, ⋯, m_{2i}, ⋯, m_{j1}, m_{j2},⋯, m_{ji}, σ_1, σ_2, ⋯, σ_j)

```
# 生成基因向量
def initWeight(dim_i, group, node_j):
    genDim = 1 + node_j + (dim_i * node_j) + node_j      # 基因向量維度
    weight1 = np.random.rand(group, genDim - node_j)* 2 - 1      # theta,w,m 值域-1~1
    weight2 = np.random.rand(group, node_j)      # sigma(σ) 值域0~1(Gussian)
    weight = np.hstack((weight1, weight2))
    return weight
```

2. RBFN 模型求取適應函數

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

$$= \sum_{j=0}^{J} w_j \varphi_j(\underline{x})$$



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

圖 1：放射性基底函數網路的架構。

i. 將 group 數量之基因向量首先一以下公式算出 $\varphi_j(\underline{x})$，再和 w 相乘加 theta 求出適應函數。

ii. 因輸出超出 1～-1 之間，使用 tanh(x) 將超過 1 的值回歸至 1 以內，而超過-1 歸到-1 以內。

iii. 將所有 group 數量算出之適應函數存入 gitFunc 陣列以便待會基因演算法使用。

```python
# RBFN求適應函數
def RBFN(data, weight, group, node, dim):
    fitFunc = np.zeros(group)    # 適應函數值有族群數個
    baseFunc = np.zeros(node)
    result = []     # TODO 取出最好的結果，供測試軌道與比較loss function
    for g in range (group):
        objTemp = 0
        for i in range(len(data)):  # 算
            for j in range(node):
                temp = 0
                for k in range(dim):
                    temp += (data[i][k]
                baseFunc[j] = math.exp(temp / ((-2) * ((weight[g][j - node]) ** 2)))  #再除該nodej對應 σ ^ 2 * (-2)
            objFunc = np.sum(baseFunc * weight[g][1 : 1 + node]) + weight[g][0] #F(x)= sum w*σ + theta

            result.append(math.tanh(objFunc))    # 將所有group的適應函數存起來
            objTemp += (data[i][-1] - math.tanh(objFunc)) ** 2      # tanh解決超過1 -1 的問題
        objTemp /= 2    # E(n) = 1/2*sum(yn-F(x))uj
        fitFunc[g] = 1 / objTemp    # 適應函數越小越佳，於是求倒數
    obj = 1 / fitFunc
    minIndex = np.argmin(obj)   # 取出最低的適應函數值
    result = result[minIndex * len(data) : (minIndex + 1) * len(data)]

    for i in range(len(result)):
        result[i] = result[i] * 40

    return fitFunc, result, minIndex
```

temp += (data[i][k] - weight[g][1 + node + j * k + k]) ** 2 #該nodej對 Xik 各維計算 (Xik - mkj) ^ 2 後相加
baseFunc[j] = math.exp(temp / ((-2) * ((weight[g][j - node]) ** 2))) #再除該nodej對應 σ ^ 2 * (-2)
temp: int
temp: int

3. 基因演算法

i. **輪盤式選擇複製**：依每個基因向量的適應函數值的大小乘上總基因數來決定被挑選至交配池的數量。

交配池總數為原先設定的族群數，若複製的數量大於此，則依權重比最低的基因開始做刪減，但若複製的數量不足，則將權重比例最高的基因，依不足的數量增加基因進交配池。

```python
# 實數型基因演算法
def geneAlgo(fitFuncO, mutationP, crossoverP, weight, node, group, minIndex): #實數型基因演算法
    #--------------------------------------
    # Reproduction  (因已算出適應函數數值，複製採用輪盤式選擇)
    bestWei = weight[minIndex]      # 取出最好的weight
    fitFunc = np.copy(fitFuncO)
    fitSum = np.sum(fitFunc)
    for i in range (len(fitFunc)):
        fitFunc[i] = np.around((fitFunc[i] / fitSum) * group)   # 求該基因在全部佔的比例 * 總數 = 被複製數
    fitRank = np.argsort(fitFunc)   # argsort求排序的對應index (小到大)

    ## 處理依比例四捨五入算基因數量，會有比原欲取族群多或少一點(e個)的問題
    e = np.sum(fitFunc) - group
    count = 0
    if e > 0:   # 將最小"非零"的e個基因扣掉
        for i in range(len(fitRank)):
            if fitFunc[fitRank[i]] == 0:    # 最小但為零不取，因為在fitFunc沒佔任何比例個
                pass
            else:
                fitFunc[fitRank[i]] -= 1
                count += 1
                if count == e:
                    break
    elif e < 0:     #   將最大的e個基因補足
        for i in range(int(-e)):
            fitFunc[fitRank[-i]] += 1

    ## 計算複製後對應之基因向量weight
    for j in range(group):
        if fitFunc[j] != 0 :
            for i in range(int(fitFunc[j])):     # 幾個基因
                try:
                    newWeight = np.append(newWeight, weight[j], axis=0)
                except:     # newWeight還未定義時跑這裡
                    newWeight = weight[j]
    newWeight = np.reshape(newWeight, (group, -1))  # 重整矩陣維度為 group * genDim
```

ii. **實數型交配**：依據使用者輸入之交配率決定交配數量，若數量不為偶數則加一，並依據下列距離控制公式進行。

- **實數型的交配過程**：交配的過程會使得兩個物種之間的距離變得更近或更遠。

$$\begin{cases} x_1' = x_1 + \sigma(x_1 - x_2) \\ x_2' = x_2 - \sigma(x_1 - x_2) \end{cases} \quad \begin{cases} x_1' = x_1 + \sigma(x_2 - x_1) \\ x_2' = x_2 - \sigma(x_2 - x_1) \end{cases}$$

σ 是隨機選取的微量正實數。

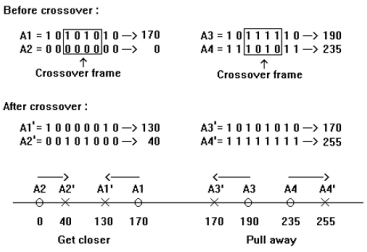


圖10.12：交配過程所造成物種移轉的情形。

```
#--Crossover--
crossNum = int(crossoverP * group)
if crossNum % 2 == 1:     # 交配數量必須為偶數
    crossNum += 1
np.random.shuffle(newWeight)
fuzzyP = 0.4
for i in range(0, crossNum, 2):      # 每次取兩個交配
    fuzzy = fuzzyP * (newWeight[i] - newWeight[i+1])
    # ---實數型交配公式： x1' = x1 + σ(x2 - x1) # x2' = x1 - σ(x2 - x1)
    newWeight[i], newWeight[i+1] = newWeight[i] + fuzzy ,newWeight[i+1] - fuzzy
```

iii. **實數型突變**：依據使用者輸入之突變率決定突變數量，隨機選取加入微量雜訊。

$$x = x + s \times random\_noise$$

其中 s 控制所加入雜訊之大小。

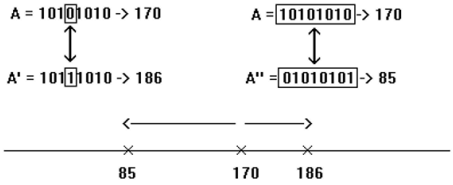


圖10.13：突變過程所造成物種移轉的情形。

```
#--Mutation--
MutaNum = int(mutationP * group)
a = np.random.rand(MutaNum, len(newWeight[0]))* 2 - 1
#為了使後面newWeight += a加雜訊的突變越來越大 *2-1 使值域變成-1~1 有加有減

a[a > mutationP] = 0     # 隨機取後大於突變機率的"位置" 的值改為0
a[a < -mutationP] = 0
a *= 0.2     # 讓值變小
b = np.zeros((group - MutaNum, len(newWeight[0])))
a = np.vstack((a, b))    # 把維度補成跟weight一樣 才能和weight運算
newWeight += a
newWeight[-1] = bestWei      # 最後一項更改為最好的weight

return newWeight, bestWei
```

4. 車子行走軌道

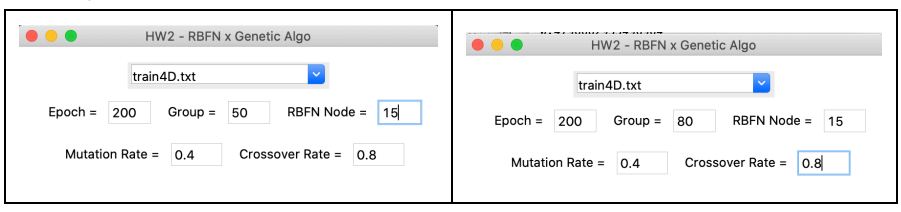
每次結束迭代，會將此輪最好的基因組送入軌道測試，若能成功走出則結束所有計算，若否則記下此輪最好的基因組（因基因演算法並不保證每輪皆能生出更好的基因組，故下輪迭代完後須與前組做比較，有更好的基因組才送軌道測試，以提高計算效能。

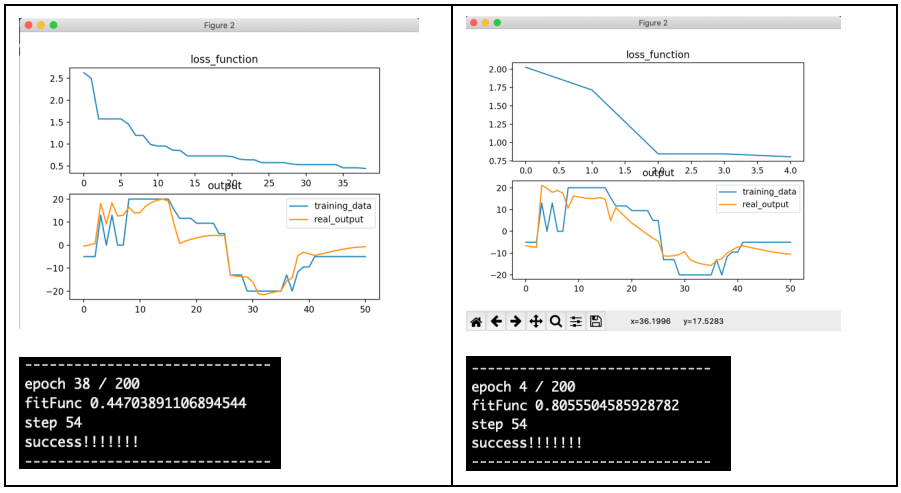
```
#Training by RBFN to get Fitness fuction and get best weight by Genetic Algo
for i in range (epoch):
    print('epoch {0} / {1}'.format(i, epoch))
    fitFunc, result, minIndex = RBFN(data, weight, group, node, dim - 1)
    weight, bestWei = geneAlgo(fitFunc, mutationP, crossoverP, weight, node, group, minIndex)

    y.append(min(1 / fitFunc))
    print('fitFunc', min(1 / fitFunc))

    #If get the best weight,try to let the car run on the track
    try:
        if not(lastBestWei == bestWei).all():
            car = Moving(dim, node, bestWei)
            detectWall, detectFinish = car.main()
            lastBestWei = np.copy(bestWei)
    except:
        lastBestWei = np.copy(bestWei)
        detectFinish = False

    print('-' * 30)
    if detectFinish == True:
        print(lastBestWei)
        if optin_saveM == 0 :
            SaveModelParams(bestWei, node, group, dim - 1)
        trackResult = car.trackResult
        plt.figure(1)
        plot_track(trackResult)
        break
```

D. 實驗結果與分析討論

1. 族群數越多/ 突變率越高/ Node 數越多，訓練時間越快

Ex.使用相同數據，族群數差快一倍時，訓練時間差距非常大，適應函數斜率也差很多。





2. 適應函數越低不表示越可能走出地圖。兩者並不成正比關係。(依實作結果發現，當適應函數很低時可能卻比起高者無法走出軌道。)
3. 6D 的資料集通常較 4D 訓練時間更久。

4. 網路有足夠複雜度而能更接近原本函數，並且搭配高突變率可以使基因有更多變化，進而可得到不錯的訓練速度與結果，提高訓練效率。嘗試後發現，將族群數設在約 70~150，突變率 0.3～0.5，Node 數 10~25 可得到不錯的效果。