類神經網路 LAB2 自駕車

113522053 蔡尚融

目錄

實作 MLP 多層感知機來進行預測使模擬車能夠自行在蜿蜒的道路中抵達終點。

一. 程式介面介紹

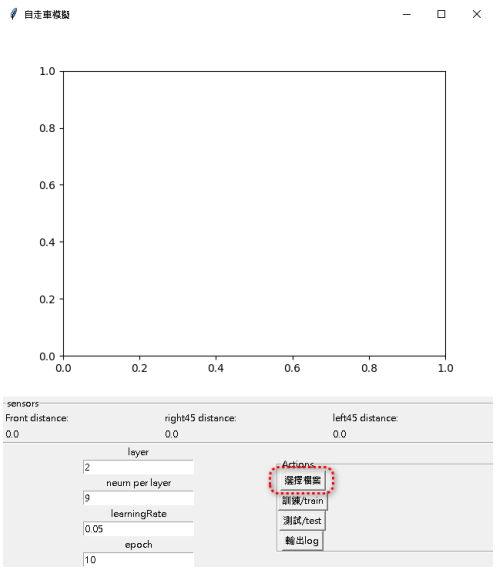
運行注意事項:

#若沒有選擇檔案將從預設路徑./input/train4dAll.txt 讀取檔案，可能會發生錯誤，建議手動進行選取。

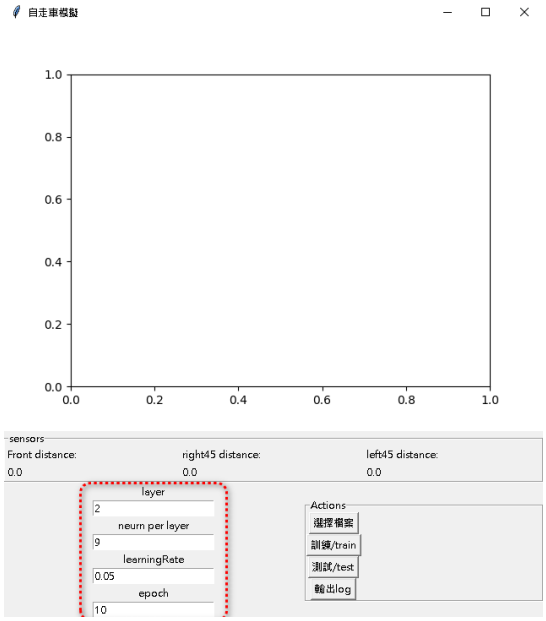
#請將軌道座標點檔案按置於與執行檔相同目錄底下./ 軌道座標點.txt 以讓程式能夠正常讀取繪製圖形

操作說明

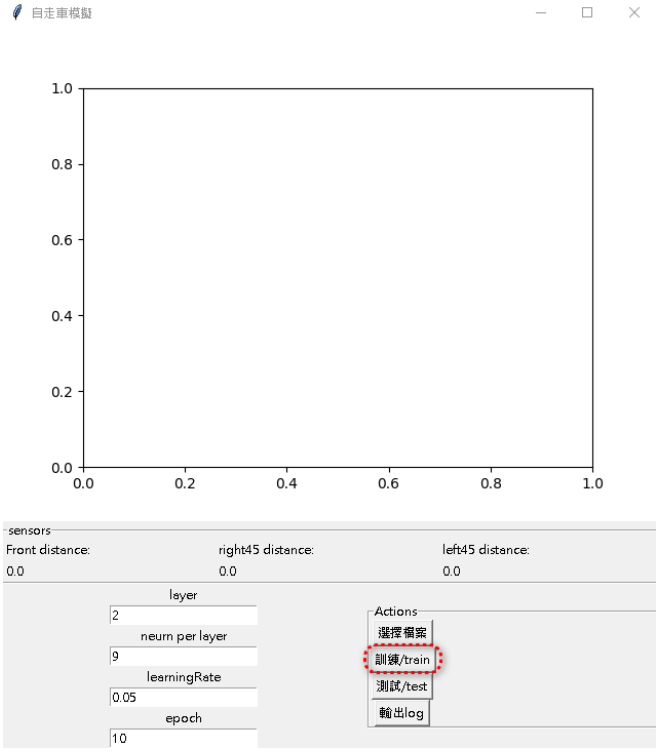
使用者首先點擊選擇檔案部分來選取訓練資料



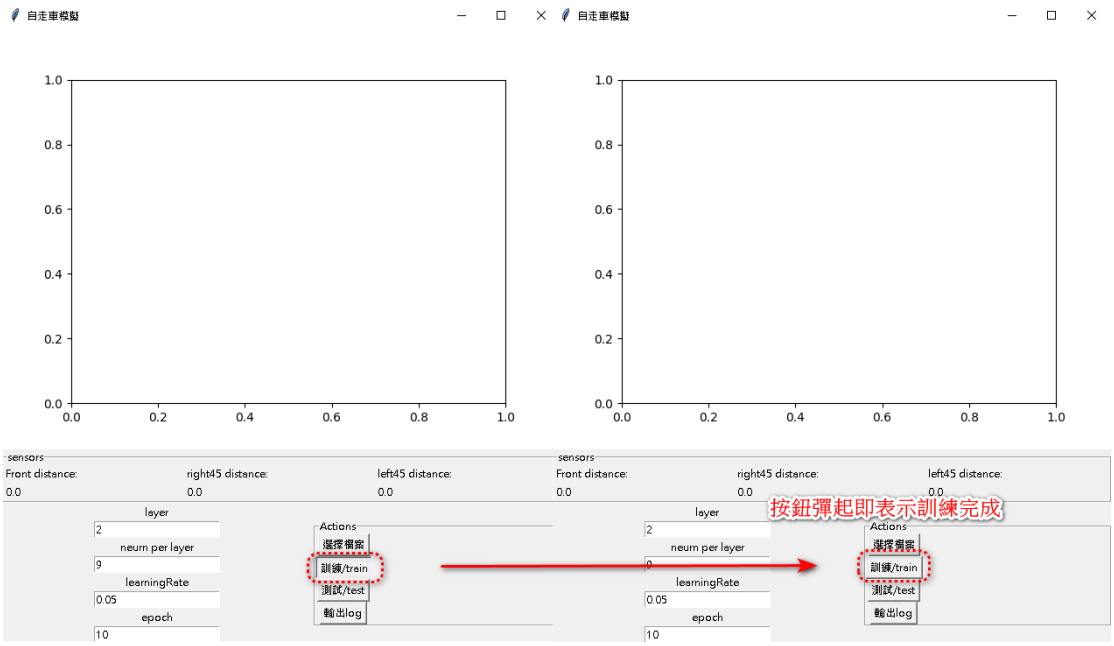
使用者可於下方的輸入框中填寫包括 MLP 網路層數(layer)、每層隱藏層所包含的神經元數量(neurn per layer)、學習率(learningRate)、迭代數(epoch)等資訊來調整使用的網路架構及訓練時的參數。



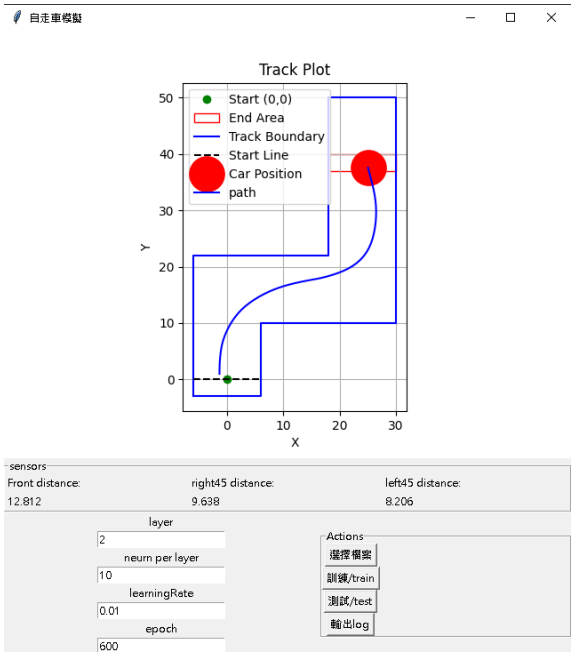
接著使用者點擊 ”訓練/train” 按鈕來開始進行 NLP 模型訓練。使用者也可以透過點擊此按鈕隨時進行重新訓練。



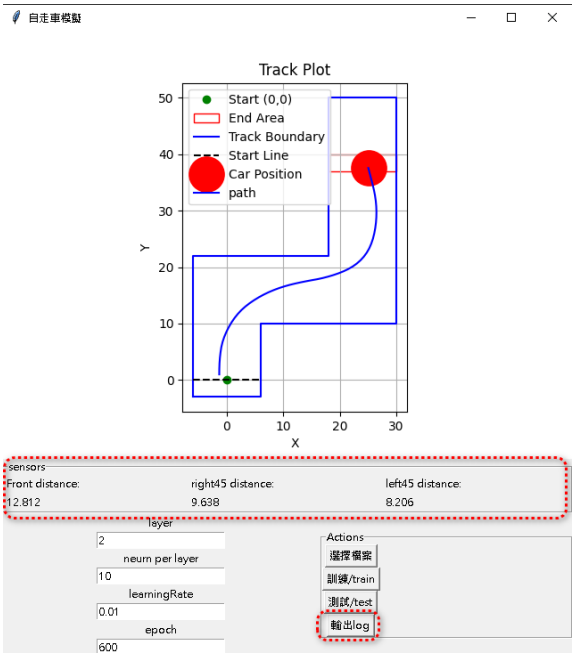
訓練過程中”訓練/train”按鈕會呈現被案下的狀態直至訓練完成。



在訓練完成後使用者可以透過點擊”測試/test”按鈕使用訓練完的模型來預測出自走車的轉向角度並在上方的繪圖欄位中動態顯示



最下方提供了輸出 log 的功能可以輸出各項感測器偵測到的距離資料角度以及模型車的轉向角度等資訊



二. 程式介紹

MLP.py

實作 MLP Class 。

```
class MLP:
    layer =2
    neurons_per_layer = 3
    input_dimension = 9
    learning_rate = 0.1
    epoch = 100
    ws = []

    def __init__(self,layer = 3,neurons_per_layer =9,learning_rate = 0.1,input_dimension = 4,epoch = 100):
        self.layer =layer
        self.neurons_per_layer = neurons_per_layer
        self.input_dimension = input_dimension
        self.learning_rate = learning_rate
        self.epoch = epoch
        self.ws = []
```

MLP Class 包含以下 funciton:

__init__()

可以於初始化時設定 MLP 層數(layer)、每層隱藏層所含的神經元數(neurons_per_layer)、learning rate、epoch 等資訊。

```
def __init__(self,layer = 3,neurons_per_layer =9,learning_rate = 0.1,input_dimension = 4,epoch = 100):
    self.layer =layer
    self.neurons_per_layer = neurons_per_layer
    self.input_dimension = input_dimension
    self.learning_rate = learning_rate
    self.epoch = epoch
    self.ws = []
```

init_ws()

用於隨機初始化鍵結值矩陣，會根據使用者輸入的 NLP 層數、每個隱藏層中的神經元數目來產生出相對應數量的鍵結值矩陣，這些鍵結值將初始到-0.01~0.01 間(上方為測試用鍵結值選項)。

```
def init_ws(self,test=False):
    if test == True:
        self.ws = [[[[-1.2],
            [1],
            [1]],[[0.3],
                  [1],
                  [1]]],
            [[[0.5],
            [0.4],
            [0.8]]]]
    else:
        first_layer = True
        for layer in range(self.layer):
            w=[]
            #如果是最後一層則只產生一組鍵結值
            for i in range(self.neurons_per_layer if layer != self.layer-1 else 1):
                w.append(np.random.uniform(-0.01,0.01,size=(self.input_dimension+1 if first_layer==1 else self.neurons_per_layer+1,1)))#如果是第一層則產生input_dimension+1
            self.ws.append(w)
            first_layer=False
```

neuron_forward()

於 MLP 進行 predict 或 trainging 時被呼叫，計算 MLP 的 forwarding 步驟，激活函數使用 sigmoid。

```
def neuron_forward(self,input,layer,index):
    input_temp = input.copy()
    input_temp.insert(0,-1)
    #print(self.ws[layer][index])
    #print(f"L={layer}I={index}")
    v = np.dot(input_temp,self.ws[layer][index])
    #print(v)
    y = 1 / (1 + np.exp(-v))
    #print(y)
    return y
```

neuron_backpropagation()

負責模型的倒傳遞及鍵結值修正工作，會根據神經元位於輸出層或是隱藏層使用不同的公式計算 w 修正量。於 calculation()中被呼叫。

```
def neuron_backpropagation(self,ylist,ans,input_data):
    deltas= []
    for layer_index in range(self.layer-1,-1,-1):
        delta = []
        for n_index in range(self.neurons_per_layer-1,-1,-1)if layer_index!=self.layer-1 else range(1):
            yi = input_data if layer_index -1<0 else ylist[layer_index-1]
            yi = yi.copy()
            yi.insert(0,-1)
            o = ylist[layer_index][n_index]
            d=0
            if layer_index == self.layer-1:
                d = ((ans-o)*o*(1-o))
            else:
                delta_weight_sum=0
                for i,dt in enumerate(deltas[0]):
                    delta_weight_sum += dt * self.ws[layer_index+1][-1-i][n_index][0]

                d = o*(1-o)*(delta_weight_sum)
            #print("yi:",yi)
            w_alt = [[self.learning_rate * d *x for x in yi]]
            #print("w:",self.ws[layer_index][n_index])
            #print("w_alt:",w_alt)
            self.ws[layer_index][n_index] += np.transpose(w_alt)
            #print("new_w:",self.ws[layer_index][n_index])

            delta.append(d)
        deltas.insert(0,delta)
```

train()

此函式會由輸入的 file path 參數來讀取檔案，並開始使用這些數據進行訓練模型的工作(呼叫 calculation()函式)。

```
86      def train(self,data_path):
87          data_path = os.path.join(data_path)
88          datas = []
89          ans = []
90          with open(data_path,"r") as file:
91              for line in file.readlines():
92                  val = line.strip().split(" ")
93                  val = list(map(lambda x:float(x),val))
94                  datas.append(val[:-1])
95
96                  ans.append(val[-1])
97              #print(ans)
98              #print(datas)
99
100         ans = list(map(lambda x:(x + 40) / 80,ans))
101         #print(ans)
102         self.input_dimension = len(datas[0])
103         self.init_ws()
104         #print(len(datas))
105         for e in range(self.epoch):
106
107             for i,data in enumerate(datas):
108                 #print(i)
109                 #print(data)
110                 self.calculation(data,True,ans[i])
111
112
113         return self.ws
```

predict()

使用者透過呼叫此函式來使用模型進行預測。此函式會呼叫 calculation()進行計算。

```
def predict(self,input):
    self.input_dimension = len(input)
    result = self.calculation(input)
    print("nr=",result[-1][0])
    return (result[-1][0] * 80)-40
```

calculation()

呼叫 neuron_forward()進行 forwarding 計算並且透過輸入的 training 參數來判斷目前是否處於訓練模式，若處於訓練模式的話則呼叫將 forwarding 結果、該次預測正確答案(ans)、該次預測的 input(input)作為參數傳入 backpropagation 來計算到傳遞修正量並進行修正。

```
def calculation(self,input,training = False,ans =""):
    y_list=[]
    first_layer =True
    for layer_index in range(self.layer):
        lr=[]
        for n_index in range(self.neurons_per_layer if layer_index!=self.layer-1 else 1):
            nr = self.neuron_forward(input if first_layer else y_list[-1],layer_index,n_index)
            lr.append(nr[0])
        y_list.append(lr)
        #print(lr)
        first_layer=False

    if training:
        self.neuron_backpropagation(y_list,ans,input)
    return y_list
```

Main.py

主要實做了繪圖以及 GUI 的部分

使用者點擊”訓練/train” button 會建立一個新的 MLP Class instance，對其初始化後使用”選擇檔案”功能選擇的檔案或預設路徑的檔案進行 MLP 訓練。

繪圖部分

Class plot:

__init__()

初始化 Class 內參數，並在使用者建立 Class plot 的 instance 時即開始繪圖。

read_coordinates()

在 plot function 中被呼叫，用於讀取軌道座標檔案內容。

```
car_positions =[]
def __init__(self,poss,state_logs):
    self.car_positions = poss
    self.plot(poss,state_logs)
    pass


def read_coordinates(self,file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

    start = tuple(map(float, lines[0].strip().split(',')))
    end_top_left = tuple(map(float, lines[1].strip().split(',')))
    end_bottom_right = tuple(map(float, lines[2].strip().split(',')))
    boundary_points = [tuple(map(float, line.strip().split(','))) for line in lines[3:]]

    return start, end_top_left, end_bottom_right, boundary_points
```

Plot_track()

繪製並動態更新自走車的移動以及路徑，動態更新主要透過 function 內的子 function : update()來完成，將 update 作為參數放入 animation.FuncAnimation()讓圖表會根據每一幀的 car_positions、state_logs 中的紀錄來進行繪製，透過這些幀的跟新就實現了動畫的效果。

```
def plot_track(self,start, end_top_left, end_bottom_right, boundary_points, car_positions,state_logs):
    ax.plot(start[0], start[1], 'go', label='Start (0,0)')

    rect = plt.Rectangle(end_top_left, end_bottom_right[0] - end_top_left[0], end_bottom_right[1] - end_top_left[1], linewidth=1, edgecolor='r', facecolor='none', label='
    ax.add_patch(rect)

    #邊界
    boundary_points.append(boundary_points[0])
    boundary_x, boundary_y = zip(*boundary_points)
    ax.plot(boundary_x, boundary_y, 'b-', label='Track Boundary')

    ax.plot([-6, 6], [0, 0], 'k--', label='Start Line')

    # 自走車與軌跡線段
    car_line, = ax.plot([], [], 'ro-', label='Car Position',markersize=np.pi*3**2)
    path_line, = ax.plot([], [], 'b', label='path')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend(loc='upper left')
    ax.set_aspect('equal', 'box')
    plt.title('Track Plot')
    plt.grid(True)
    def init():
        car_line.set_data([], [])
        path_line.set_data([],[])
        return car_line,

    def update(frame):
        if frame < len(car_positions):
            #car_x, car_y = zip(*car_positions[:frame+1])
            car_x, car_y = car_positions[frame]
            path_x, path_y = zip(*car_positions[:frame+1])

            car_line.set_data([car_x], [car_y])
            path_line.set_data(path_x,path_y)
            FD.set(round(state_logs[frame][0],3))
            RD.set(round(state_logs[frame][1],3))
            LD.set(round(state_logs[frame][2],3))
        return car_line,path_line,
    ani = animation.FuncAnimation(fig, update, frames=len(car_positions), init_func=init, blit=True, repeat=False)
    #plt.show()
    canvas.draw()
```

Plot()

在 class plot 的 instace 被建立後__init__()會呼叫此 function，並透過此 function 分別呼叫 read_coordinates()、plot_track()來開始進行繪圖

```
def plot(self,car_positions,state_logs):
    # 開始繪圖
    ax.cla()
    file_path = '軌道座標點.txt'
    start, end_top_left, end_bottom_right, boundary_points = self.read_coordinates(file_path)

    self.plot_track(start, end_top_left, end_bottom_right, boundary_points, car_positions,state_logs)
    time.sleep(0.3)
```

按鈕 function 部分

Check_Dimension()

主要在讀取或訓練時檢查輸入資料的為度藉此調整第一層隱藏神經元的鍵結值數量。

Select():

當使用者點擊”選擇檔案”時被呼叫。

```
def check_Dimension(filepath):
    global D6D
    D6D=False
    with open(filepath,"r") as file:
        line = file.readline()
        if len(line.split(" "))==6:
            D6D=True
    print(f"D6D={D6D}")


#選擇檔案按鈕
def select():
    FP.set(filedialog.askopenfilename())
    print("reading data from:",FP.get())
    check_Dimension(FP.get())
```

Training()

當使用者點擊”訓練/train”按鈕時被呼叫，建立 MLP instance 並開始訓練。

```
#訓練按鈕
def training():
    global MLP1
    filepath = FP.get()
    check_Dimension(filepath)
    MLP1 = MLP.MLP(learning_rate=LR.get(),epoch=EP.get(),layer=LAYER.get(),neurons_per_layer=NR.get())

    MLP1.train(filepath)
```

Testing()

當使用者點擊”測試/test”按鈕時被呼叫。呼叫 run_example 模擬自駕車後，建立 plot instance 已開始繪圖。

```
#測試按鈕
def testing():
    global poss,state_logs,action_logs
    poss,state_logs,action_logs = run_example(MLP1,D6D)
    plot(poss,state_logs)
    canvas.draw()
```

Log_output()

根據資料維度，於當前目錄下建立路徑與車輛狀態、轉向角度的輸出 log。

```
def log_output():
    if D6D:
        with open("track6D.txt", 'w') as file:
            for i in range(len(poss)):
                line = f"{poss[i][0]} {poss[i][1]} {state_logs[i][0]} {state_logs[i][1]} {state_logs[i][2]} {action_logs[i]}\n"
                file.write(line)
    else:
        with open("track4D.txt", 'w') as file:
            for i in range(len(poss)):
                line = f"{state_logs[i][0]} {state_logs[i][1]} {state_logs[i][2]} {action_logs[i]}\n"
                file.write(line)
```

Simple_playground.py

使用者點擊”測試/test” button 會呼叫 simple_playground.py 中的 run_example() funciton，透過 MLP.prediction()使用訓練好的模型來預測模型車方向盤角度，並以圖表的方式呈現出模型車、移動軌跡、感測器數值等。

主要修改了 predictAction() function，新增了傳入參數 model，並於 function 內透過由 Main.py 中 predict function 傳入的 MLP instance 來根據當前的 state 預測模型車方向盤應轉角度。

predictAction()

```
    def predictAction(self, state,model):
        '''
        此function為模擬時，給予車子隨機數字讓其走動。
        不需使用此function。
        '''
        #r.randint(0, self.n_actions-1)
        print(model.predict(state))
        return model.predict(state)
```

另外修改了 calWheelAngleFromAction function 讓 action 直接等於 angle 進行回傳:

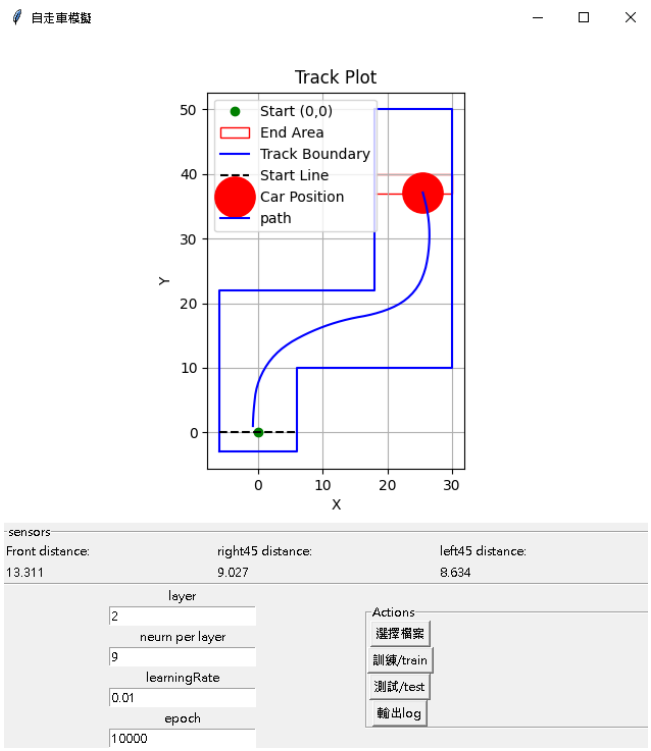
calWheelAngleFromAction()

```
def calWheelAngleFromAction(self, action):
    """ angle = self.car.wheel_min + \
        action*(self.car.wheel_max-self.car.wheel_min) / \
        (self.n_actions-1) """
    #print(f"angle1 = {angle}")
    angle = action
    return angle
```

三. 實驗結果

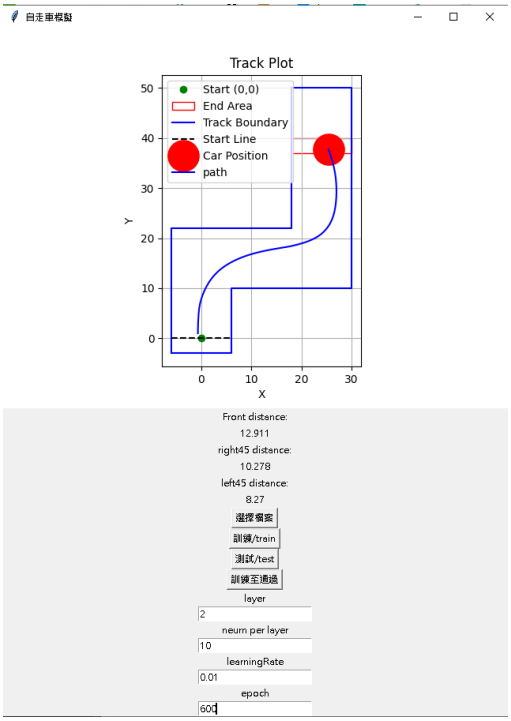
train4dAll

右圖為使用 train4dAll 資料集作為訓練資料，達成走到終點這個目標的模型參數以及具體的路線圖。使用了一層隱藏層加上一層輸出層的兩層 MLP 網路架構，隱藏層中總共有 9 個神經元輸出層 1 個神經元。使用 0.01 的學習率在訓練了 1 萬次下得到成功走到終點的結果(因為模擬程式每次出發點似乎會有一點差異，故進行測試不一定每次都能夠走到終點，需要試幾次)



train6dAll

右圖為使用 train6dAll 資料集訓練的模型，成功到達終點的模型參數同樣使用了一層隱藏層一層輸出成的兩層 MLP 架構，但隱藏層有 10 個神經元。此模型在使用 0.01 的學習率下訓練了 600 epoch 就能成功抵達終點，可以觀察到花費的訓練次數比起四維的資料少一點。

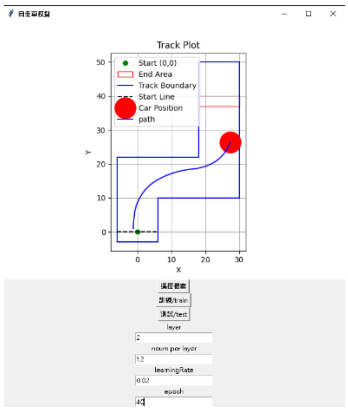


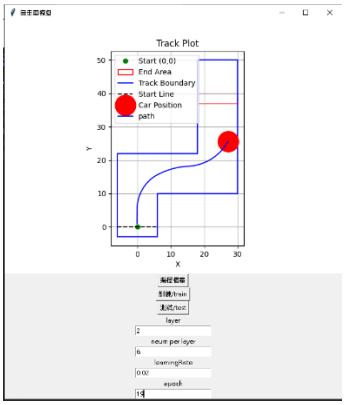
四. 分析

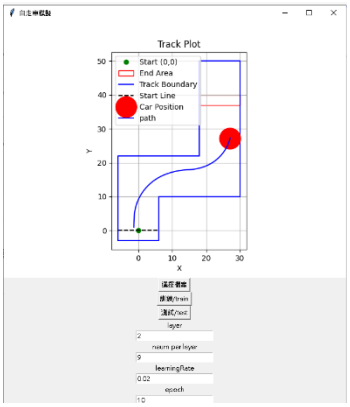
這次的實驗花了許多時間在調整訓練參數神經元層數、每層隱藏層神經元數量上，在過程中發現到透過 MLP 來訓練自走車的話預測出來的轉向角度無法達到訓練資料中最大轉向角度的-40 以及 40 度轉向，最多轉向 30 幾度或-30，進而導致在狹窄處有時候會難以成功轉過轉角而撞到牆，而增加了以 epoch 次數可以讓模型預測出的最大轉向角度些微增加，但最後還是無法達到 40 度以及-40 度的最大轉向。

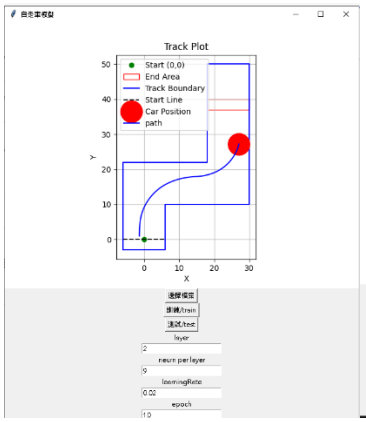
比較四維度的輸入以及六維度的輸入，可以發現使用六維度的輸入資料可以在比較少的訓練次數下就成功讓自駕車抵達終點，觀察到多提供的座標資料確實起到了作用幫助模型判別轉向的時機。

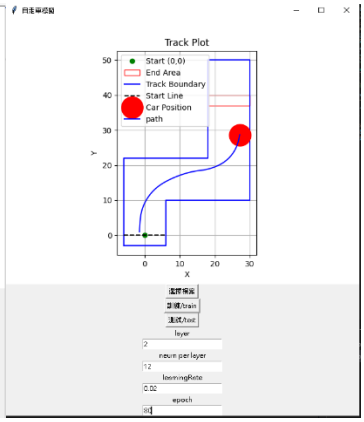
一開始在低 epoch 下進行訓練，因為得到很接近終點的結果，故花了許多時間在這些範圍內進行微調嘗試，但最後都沒能在較低 epoch 下成功走到終點，以下是一些在較低 epoch 就相當接近終點的紀錄，推測可能原因是上面提到模型擬合度不足，最大轉向輸出角度不夠才導致轉向不足的問題。

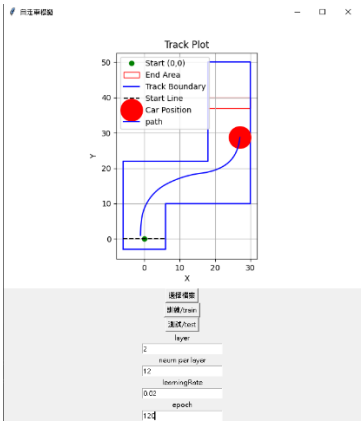

Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
layer
2
neum per layer
12
learningRate
0.02
epoch
40

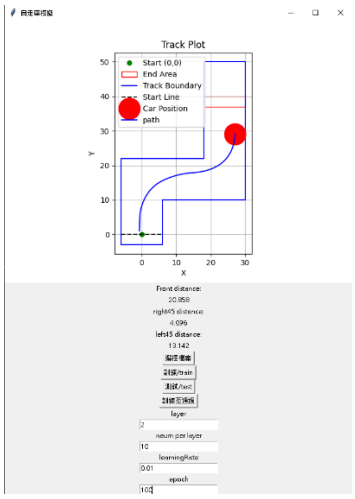

Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
layer
2
neum per layer
6
learningRate
0.02
epoch
15


Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
layer
2
neum per layer
9
learningRate
0.02
epoch


Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
layer
2
neum per layer
9
learningRate
0.02
epoch


Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
layer
2
neum per layer
12
learningRate
0.02
epoch


Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
layer
2
neum per layer
12
learningRate
0.02
epoch


Track Plot
Start (0,0)
End Area
Track Boundary
Start Line
Car Position
path
X
Y
Front distance:
10.869
right45 distance:
4.096
left45 distance
13.142
layer
2
neum per layer
10
learningRate
0.01
epoch