計算型智慧

作業三

PSO

學號：109401553

系級：資管三 B

學生：楊雲杰

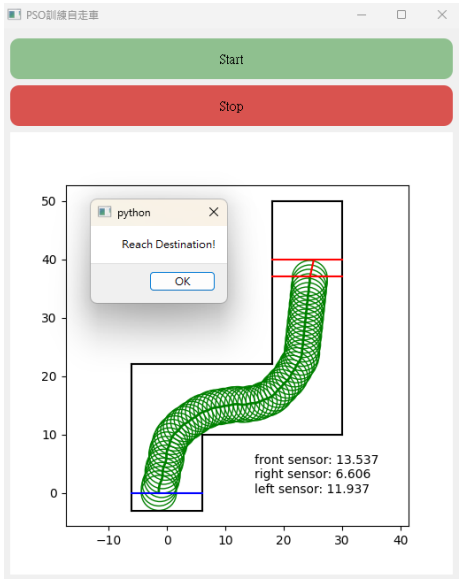
日期：2024/06/11

一、程式介面說明

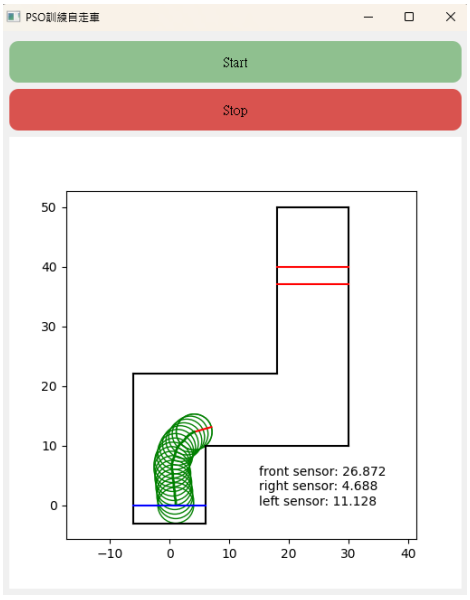
我的執行檔路徑為 exe_file/simple_playground/simple_playground

圖一、執行檔路徑

起點線為藍色線，終點為紅色長方形，車子為綠色空心圓圈(為了方便看出軌跡)。介面上方有按鈕 Start，點擊即可讓車子進行一次模擬。Stop 則是在模擬進行中可以點擊暫停觀察狀況。介面右下角另有三行文字，分別為正前方、右 45 度角、左 45 度角的感測器所測量出的距離。



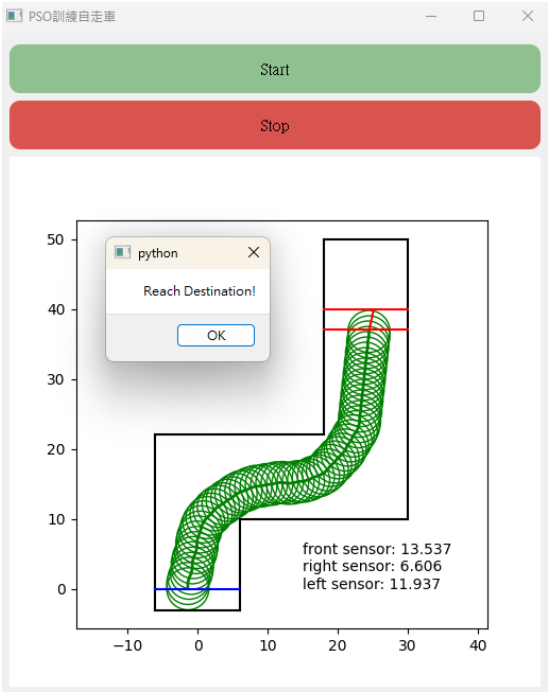
圖二、成功畫面



圖三、失敗畫面

二、實驗結果

依據題目要求，我使用左側、前方、右側三個感測器，並使用 MLP、PSO 演算法實作自走車，成功讓自走車一次抵達終點。



圖四、一次走到終點

三、PSO 實作細節

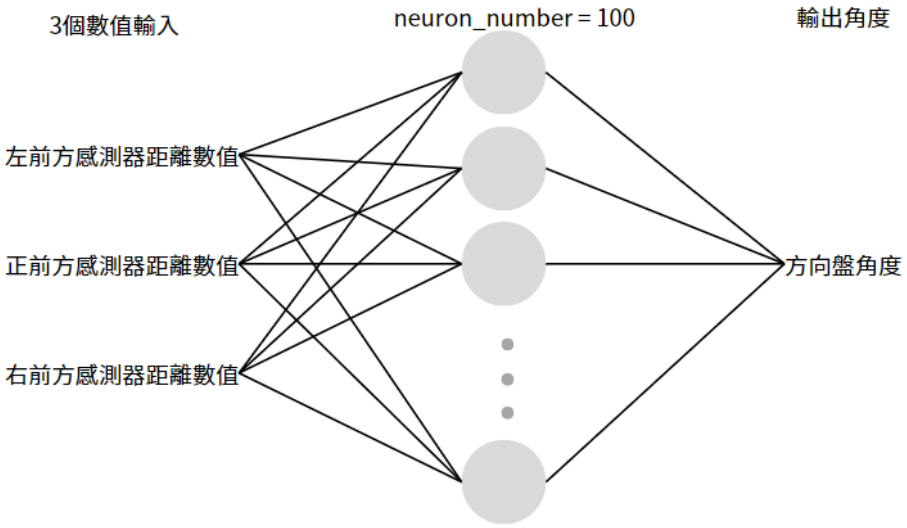
1. 參數設定

a. 粒子個數：50
b. 最大迭代次數：200
c. V(0) = 0
d. Vmax：10
e. Vmin：-10
f. $\varphi_1 = 2, \varphi_2 = 2$

2. 自走車轉向控制

我是利用 MLP 進行實作，其中 MLP 有三層，分別為一個輸入層、一個隱藏層、一個輸出層。輸入層用於輸入左前、中間、右前的感測器所偵測到的距離值，輸出層則是輸出方向盤的角度。除此之外，我的隱藏層使用的激活函數是 ReLu 函數，輸出層則是使用我自行設計的 tanh 函數。我在這個 tanh 函數中乘了 1/5，因為我想要讓-1~1 之間的過渡在更寬的範圍內發生(圖七是-40~40，因為乘了 40 倍)。而至於乘上 1/5 而

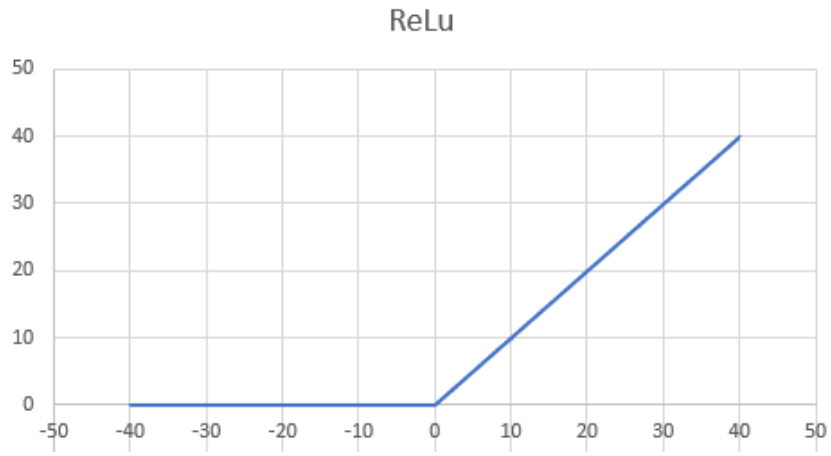
不是其他數值是因為經過測試我發現這個效果相對較好。



圖五、MLP

```
def ReLu(x):
    return np.maximum(0, x)
```

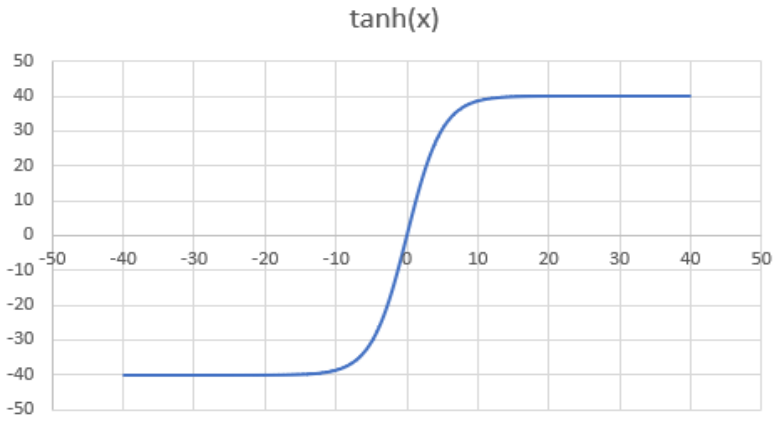
圖六、ReLu 函數程式碼



圖五、ReLu 函數

```
wheel_angle = 40 * np.tanh(SUMout * 1/5)
```

圖六、tanh 程式碼



圖七、自行設計的 tanh 函數

3. Fitness function(適應度函數)

這個適應度函數主要是在評估自走車的效果。其中，我將自走車在 x 軸上的最終位置*0.8，表示水平方向上的表現。將自走車在 y 軸上的位置*1，表示垂直方向上的表現，另外，我扣除自走車所走路徑長度*0.4，因為我希望路徑越短越好。最後如果自走車成功抵達終點給予 200 分的獎勵，失敗給予-50 分獎勵。

```python
# 適應度函數
def calculate_fitness(self, car_traj: list, succeeded: bool):
    if succeeded:
        self.find_successParticle = True
    # 考慮終點的 x, y 坐標值
    final_position_score = 0.8 * car_traj[-1][0] + 1.0 * car_traj[-1][1]
    # 路徑長度的懲罰
    length_penalty = 0.4 * len(car_traj)
    # 基本分數
    base = 200 if succeeded else -50  # 未成功則給予懲罰分
    # 總分
    total_score = base + final_position_score - length_penalty

    return total_score
```

圖八、適應度函數

4. Class Particle(粒子)

在我的實作中，每個粒子包含了 MLP 的權重函數、訓練時的軌跡、前一時間訓練速度以及當次訓練速度。

在這個類別中，我還利用 PSO 演算法更新粒子速度與位置(利用權重矩陣實現)。其中我的參數 learning rate φ_1、φ_2 設為 $\varphi_1=2, \varphi_2=2$，速度限制 Vmax = 10，Vmin = -10，V(0) = 0。

```python
# 粒子
class Particle:
    def __init__(self, neuronNumber_hid=100):
        self.neuronNumber_hid = neuronNumber_hid
        self.Whid = np.random.uniform(-8.0, 8.0, size=(3, self.neuronNumber_hid))
        self.Wout = np.random.uniform(-8.0, 8.0, size=(self.neuronNumber_hid, 1))
        self.pre_velocity = 0 #前一時間速度
        self.cur_velocity = 0 #當前速度
        self.route_history = [] #紀錄車輛行駛路徑
        self.succeeded = False #紀錄是否抵達終點
```

圖九、Particle 初始值設定

```python
# 更新權重
def update_velocity_and_position(self, previous_best, global_best, phi1=2.0, phi2=2.0):
    vmax = 10
    vmin = -vmax
    self.pre_velocity = self.cur_velocity
    # r1, r2 = np.random.rand(), np.random.rand()  原先想寫這行發現效果不彰
    #把權重堆疊成一個數組
    all = np.vstack((self.Whid, self.Wout.reshape(1, -1)))
    # 計算新的速度，包含自我認知分量和社會認知分量
    # velocity_update = self.pre_velocity + phi1 * r1 * (previous_best - all) + phi2 * r2 *(global_best - all)
    velocity_update = self.pre_velocity + phi1 * (previous_best - all) + phi2 * (global_best - all)
    self.cur_velocity = np.clip(self.pre_velocity + velocity_update, vmin, vmax) #速度限制
    #更新權重(位置)
    self.Whid += self.cur_velocity[:-1]
    self.Wout += self.cur_velocity[-1].reshape(-1, 1)
```

圖十、更新速度及位置

5. 模行訓練過程

以下是我的 PSO 訓練過程，其中，我的終止條件是找到達到目標的粒子或最大迭代次數為 200 次。另外，關於全局最佳解更新的部份，我的實作方式是檢查所有粒子，若某粒子的適應度超過當前紀錄的全局最佳適應度值，更新全局最佳解。而個體最佳解更新則是當前位置若優於個人歷史的最佳位置即進行更新。

```python
train_model(self, max_iteration = 200):
iteration_count = 0
while not self.find_success_particle and iteration_count < max_iteration:
    # 讓每一台車子跑一次
    for particle in self.particles:
        particle.run_in_playground(self.playground)

    # 計算歷史最佳和鄰近最佳
    for ind_out, particle in enumerate(self.particles):
        # 若為歷史最佳則更新
        if (self.calculate_fitness(particle.get_route(), particle.can_finish()) > self.previous_best_particle_fitnessVal):
            self.previous_best_particle_fitnessVal = self.calculate_fitness(particle.get_route(), particle.can_finish())
            self.previous_best_particle_weight = particle.get_weight()
            if (particle.can_finish()):
                self.find_success_particle = True

        # 更新鄰近最佳
        global_best_particle_weight = particle.get_weight()
        global_best_particle_fitnessVal = self.calculate_fitness(particle.get_route(), particle.can_finish())
        for ind_in, neighbor in enumerate(self.particles):
            # 若找到鄰近最佳
            if (self.calculate_fitness(neighbor.get_route(), neighbor.can_finish()) > global_best_particle_fitnessVal):
                global_best_particle_weight = neighbor.get_weight()
                global_best_particle_fitnessVal = self.calculate_fitness(neighbor.get_route(), neighbor.can_finish())
        # 更新粒子的速度和位置
        particle.update_velocity_and_position(self.previous_best_particle_weight, global_best_particle_weight)
    iteration_count += 1
```

圖十一、模型訓練過程

四、分析

在寫這次作業的過程中，MLP 設計、fitness function 的撰寫以及建立 class Particle 是我認為非常有挑戰性的部分。

一開始，我採用 RBFN 的方式建立模型。然而，不知道為何我的自走車總是會走出一些奇怪的路徑，在牆邊一直打轉。後來我果斷決定利用我更為熟悉的 MLP 來訓練模型，改變策略後，我發現即使我使用 MLP 的方法我也會遇到這種奇怪的軌跡，因此我得出的結論是我認為 PSO 的方法有時候會無法完全收斂。

第二個遇到的問題是適應度函數的設計。原先的設計是單純考量是否成功，若成功則給予 200 獎勵，失敗給予-50 獎勵，然而，這個方法效果不彰。於是我加入了考慮終點 x、y 座標值的設計以及路徑長度懲罰的設計。最後，幾經測試，我得到了不錯的結果。

第三個遇到的問題是我原先是直接利用 playground 進行實作，但是我發現我的程式常常會跑不動。後來我才發現原來是因為我在建立 particle 時會宣告非常多個 playground，造成記憶體不足。於是，我重新修改我的設計方式，在 class Particle 中寫了一個 run_in_playground 方法，幫助我不用一直重複呼叫。

總而言之，這學期我花了非常多的時間及心血，製作了三種讓自走車可以走到終點的程式。儘管做出來的程式仍有不完整之處，但在三個作業中，無論是用哪種方式訓練都讓我學習到了非常多，也讓我更精進我的程式能力。最後，希望未來我能將本學期課程所學應用在各種不同的地方。