計算型智慧

作業三

PSO

學號：109401553

系級：資管三 B

學生：楊雲杰

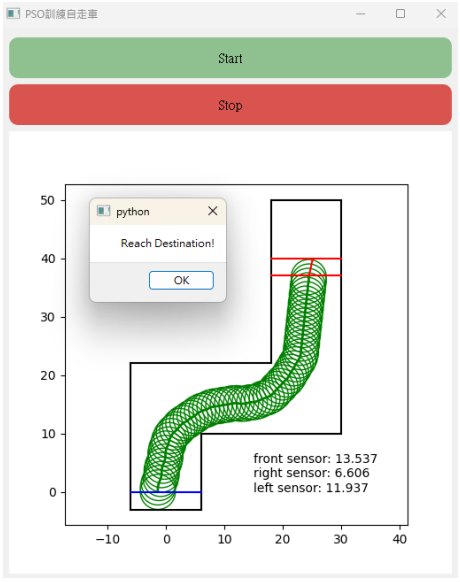
日期：2024/06/11

一、程式介面說明

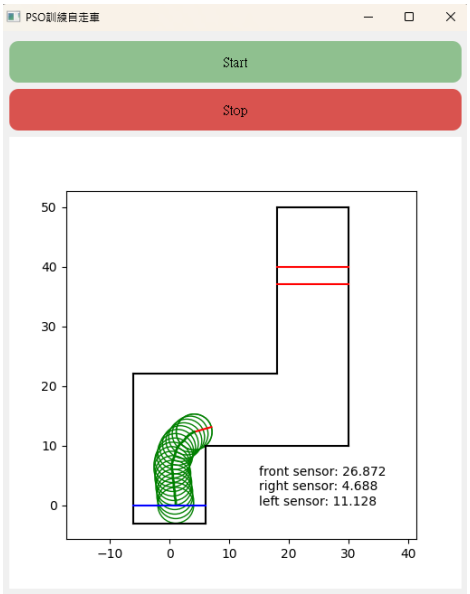
我的執行檔路徑為 exe_file/simple_playground/simple_playground

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。



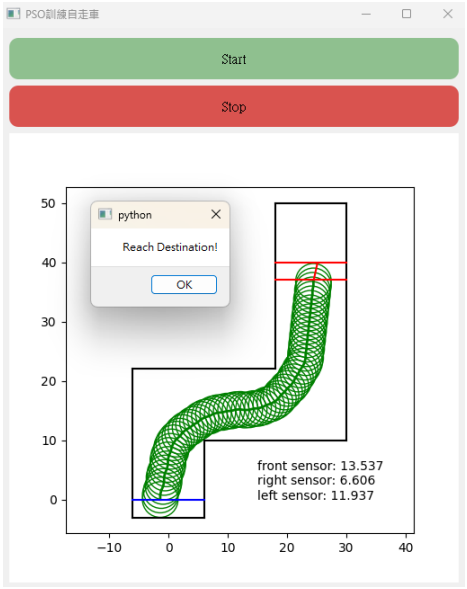
圖二、成功畫面



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。(執行時若想要觀察不同起點出發的狀況重複點擊 Start 即可，若該次訓練不理想，可以重新點擊 exe 檔重新訓練。)



圖四、一次走到終點

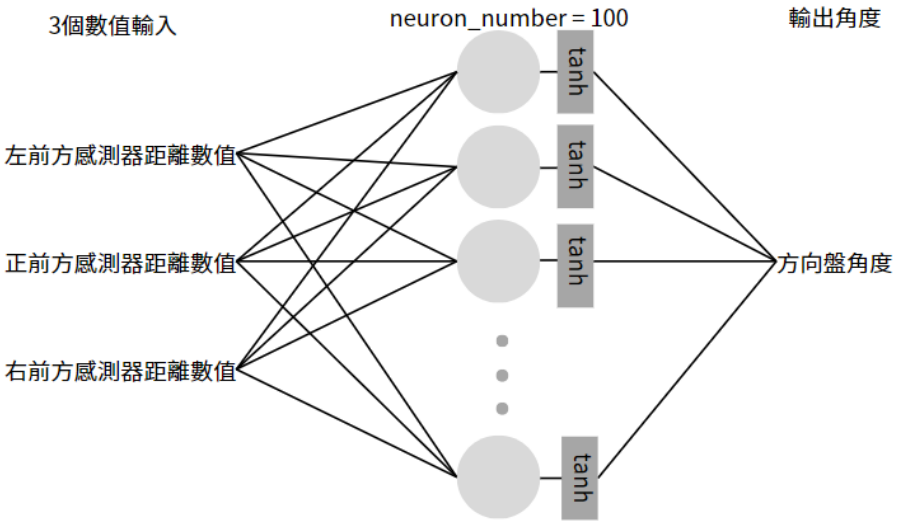
三、PSO 實作細節

1. 參數設定

a. 粒子個數：50
b. 最大迭代次數：80
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

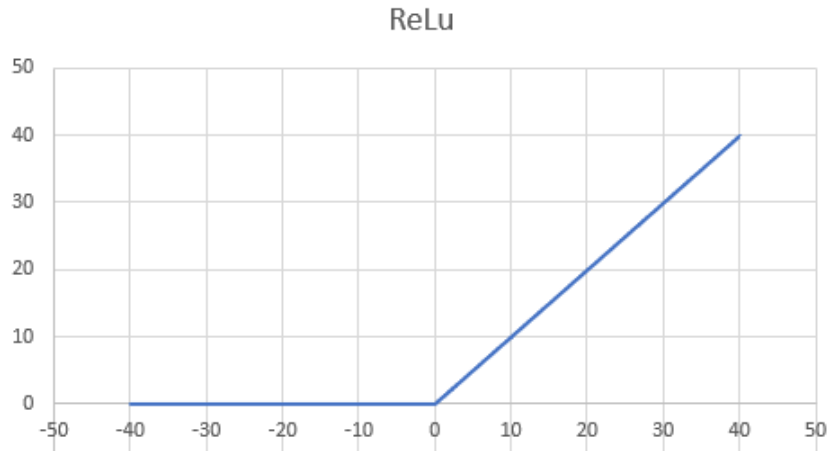
我是利用 MLP 進行實作，其中 MLP 有三層，分別為一個輸入層、一個隱藏層、一個輸出層。輸入層用於輸入左前、中間、右前的感測器所偵測到的距離值，輸出層則是輸出方向盤的角度。除此之外，我的隱藏層使用的激活函數是 ReLu 函數，輸出層則是使用我自行設計的 tanh 函數。我在這個 tanh 函數中乘了 1/4，因為我想要讓-1~1 之間的過渡在更寬的範圍內發生(圖八是-40~40，因為乘了 40 倍)。



圖五、MLP

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

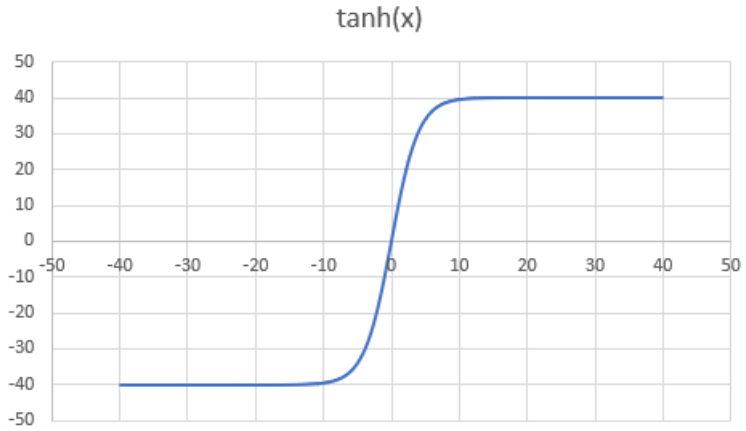
圖六、ReLu 函數程式碼



圖七、ReLu 函數

```
wheel_angle = 40 * np.tanh(SUMout * 1/4)
```

圖八、tanh 程式碼



圖九、自行設計的 tanh 函數

3. Fitness function(適應度函數)

$$total\ score = 0.8 * x\text{軸最終位置} + 2 * y\text{軸最終位置} - 0.4 * \text{自走車路徑長} + (\text{走到終點給予 200 分否則} - 50)$$

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*2，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。

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、當次訓練速度、個人最佳位置以及個人最佳適應度。

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。

```
# 更新權重
def update_velocity_and_position(self, previous_best, global_best, phi1=2, phi2=2):
    vmax = 10
    vmin = -vmax
    self.pre_velocity = self.cur_velocity
    r1, r2 = np.random.rand(), np.random.rand()

    #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)
    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

我的 PSO 訓練過程的終止條件是有粒子達到終點就終止(我在程式碼中有保留另一種方法也就是最大迭代次數(max_iteration) = 80，若助教有需要測試再移除註解即可)。這兩種方法都確保了在找到可行解或充分探索後停止。在每次迭代中，每個粒子會根據其當前位置進行模擬，並計算相應的適應度值。如果該適應度值超過該粒子迄今為止的最佳適應度，該粒子的個人最佳位置將更新為當前位置。同時，如果當前位置的適應度超過了歷史全局最佳適應度，全局最佳位置也會更新。最後，每次迭代結束時，所有粒子會根據全局最佳位置和個人最佳位置更新速度和位置。

```python
def train_model(self, max_iteration=80):
    iteration_count = 0
    global_best_fitness = float('-inf')
    global_best_weight = None

    while not self.find_success_particle:
    # while iteration_count < max_iteration:
        # 每個粒子模擬一次並計算適應度
        for particle in self.particles:
            particle.run_in_playground(self.playground)
            fitness = self.calculate_fitness(particle.get_route(), particle.can_finish())

            # 更新個人歷史最佳
            if fitness > particle.best_fitness:
                particle.best_fitness = fitness
                particle.best_weight = particle.get_weight()

            # 檢查是否到達終點
            if particle.can_finish():
                self.find_success_particle = True

            # 更新全局最佳
            if fitness > global_best_fitness:
                global_best_fitness = fitness
                global_best_weight = particle.get_weight()

        # 根據全局最佳值更新粒子速度和位置
        for particle in self.particles:
            particle.update_velocity_and_position(particle.best_weight, global_best_weight)

        iteration_count += 1

    self.previous_best_particle_weight = global_best_weight
```

圖十一、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，在牆邊一直打轉並且效果不好。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，我成功訓練出可以一次走到終點的自走車，儘管有時還是會有奇怪路徑的出現，但成功率很高。

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。