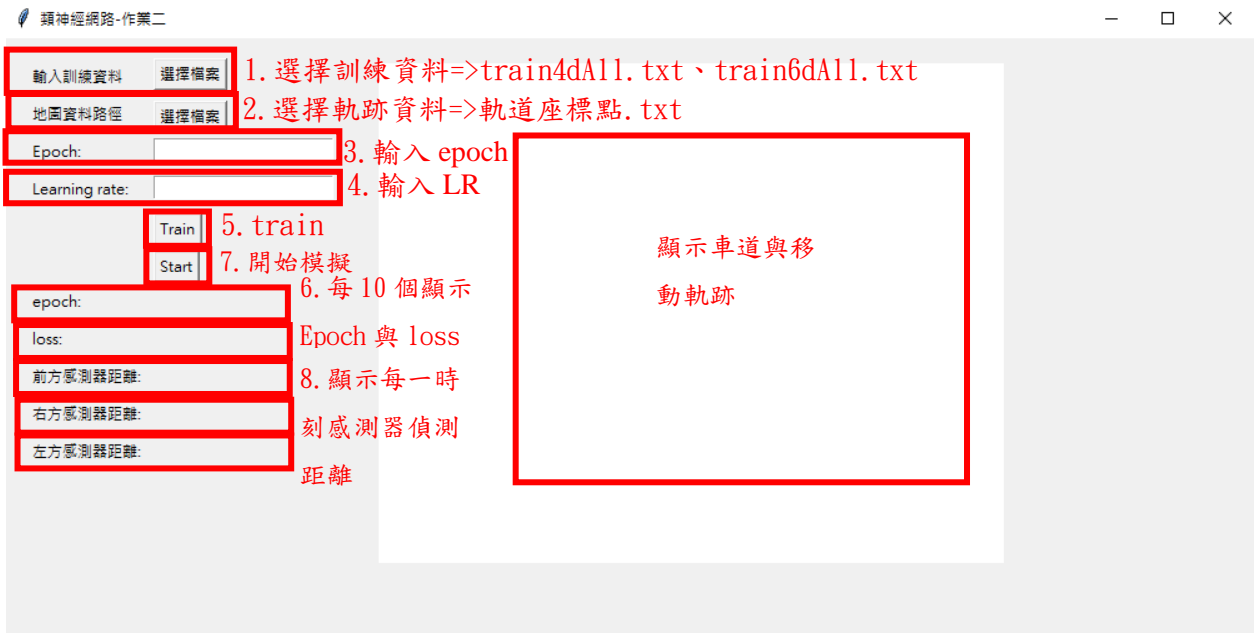


類神經網路作業二報告

(1) 程式介面與操作說明

詳細操作可以看 111522101_HW2.mkv，需要把軌跡圖清空在選一次地圖資料路徑就好。



(2) 程式碼說明

本次作業使用四個 class 和一個 main 當作主要架構，class 部份分別是 Dataprocessor、Kmeans、model、simulator，以下會分別進行講解

➤ Main.py

main.py 主要是負責介面的設計與 class 的調動，GUI 主要是使用 tkinter 進行撰寫，在 button 的部分分別會調用 get_file_url() 取得訓練檔案資料與 draw_track 繪製路徑圖

```
96     window = tk.Tk()
97
98     f = Figure(figsize=(5, 4), dpi=100)
99     plot_view = tk.Frame(window)
100     plot_view.place(x=300,y=20)
101     canvas = FigureCanvasTkAgg(f, plot_view)
102     canvas.get_tk_widget().pack(side=tk.RIGHT, expand=1)
103
104     window.geometry("1000x500+200+300")
105
106     window.title('類神經網路-作業二')
107     #選擇檔案
108     file_name = tk.StringVar() # 設定 text 為文字變數
109     file_name.set('') # 設定 text 的內容
110
111     route_name = tk.StringVar() # 設定 text 為文字變數
112     route_name.set('') # 設定 text 的內容
113
114     tk.Label(window, text='輸入訓練資料').place(x = 20,y = 20)
115
116     tk.Button(window, text='選擇檔案',command= lambda: get_file_url(file_name)).place(x = 120,y = 16)
117
118     tk.Label(window, text='地圖資料路徑').place(x = 20,y = 50)
```

Train_model()會取得學習率與 epoch 次數進行 model 的訓練

print_result()會使用調用模擬程式開始進行感測器的計算與路徑的繪製

```
148 #開始訓練資料
149 tk.Button(window, text='Train',command= lambda: train_model(
150     int(interaction.get()),
151     float(learning_rate.get()),
152     window,
153     epoch,
154     loss
155 ).place(x = 120,y = 140)
156
157 front_distance = tk.StringVar()
158 front_distance.set('')
159 tk.Label(window, text='前方感測器距離:').place(x = 20,y = 260)
160 tk.Label(window, textvariable=front_distance).place(x=120, y=260)
161
162 right_distance = tk.StringVar()
163 right_distance.set('')
164 tk.Label(window, text='右方感測器距離:').place(x = 20,y = 290)
165 tk.Label(window, textvariable=right_distance).place(x=120, y=290)
166
167 left_distance = tk.StringVar()
168 left_distance.set('')
169 tk.Label(window, text='左方感測器距離:').place(x = 20,y = 320)
170 tk.Label(window, textvariable=left_distance).place(x=120, y=320)
171
172
173 tk.Button(window, text='Start', command= lambda: print_result(window,canvas,front_distance,right_distance,left_distance)).place(x = 120,y = 170)

52 def train_model(epochs,learning_rate>window,epoch,loss):
53     global rbfnModel,feature_len,y_train
54     dataprocessor = Dataprocessor()
55     x_train,y_train = dataprocessor.splitFile(file_url)
56     feature_len = len(x_train[0])
57     kmeans = KMeans(x_train,len(x_train[0]))
58     m,sigma = kmeans.process()
59     print(m,sigma)
60
61     rbfnModel = Model(epochs,learning_rate,m,sigma,len(x_train[0]))
62     rbfnModel.train(x_train,y_train>window,epoch,loss)
63
64 def print_result(window,canvas,front_distance,right_distance,left_distance):
65     simu = Simulator(0,0,90)
66     four_dimension,six_dimension = simu.start>window,canvas,f_plot,feature_len,rbfnModel,front_distance,right_distance,left_distance)
67     if feature_len == 3:
68         save_4d_result(four_dimension)
69     else:
70         save_6d_result(six_dimension)
71
72 def save_4d_result(four_dimension):
73     file = open("track4D.txt", "w")
74     for i in range(len(four_dimension)):
75         towrite = ' '.join(str(item) for item in four_dimension[i])
76         file.write(f"{towrite}\n")
77     file.close
78
79 def save_6d_result(six_dimension):
80     file = open("track6D.txt", "w")
81     for i in range(len(six_dimension)):
82         towrite = ' '.join(str(item) for item in six_dimension[i])
83         file.write(f"{towrite}\n")
84     file.close
85
```

使用 Dataprocessor 呼叫將資料分成 input 與 label

使用 Kmeans 挑出 rbfn 的初始值 m 跟 sigma

將資料丟進 RBFN model 進行訓練

將訓練好的 model 與 GUI 所需設定的物件放入 simulator，並取得路徑與方向盤資訊

看維度繪製檔案

➤ Dataprocessor

將資料分成 input 跟 label，會根據輸入的維度不同做不同處理，並且會將 label 使用 min max scaler 正規化，以便後續 model 訓練。

```
3 class Dataprocessor:
4     def __init__(self):
5         self.x_train = []
6         self.y_train = []
7
8     def splitFile(self, dataset_url):
9         with open(dataset_url, 'r', encoding='utf-8') as f:
10             for data in f.readlines():
11                 data = data.split(' ')
12                 if (len(data) == 4):
13                     self.x_train.append([float(data[0]), float(data[1]), float(data[2])])
14                     self.y_train.append(float(data[3]))
15                 else:
16                     self.x_train.append([float(data[0]), float(data[1]), float(data[2]), float(data[3]), float(data[4])])
17                     self.y_train.append(float(data[5]))
18
19             self.x_train = np.array(self.x_train)
20             self.y_train = np.array(self.y_train)
21             self.y_train = (self.y_train - np.amin(self.y_train)) / (np.amax(self.y_train) - np.amin(self.y_train))
22         f.close()
23
24     return self.x_train, self.y_train
```

讀檔與將資料從 string 轉成 float

```
26 def readfile(url):
27     read = open(url)
28     file = read.readlines()
29     read.close()
30     return file
31
32 def text_to_numlist(dataset):
33     """load text dataset to numerical list dataset
34
35     Args:
36         dataset (string): txt or other file
37
38     Returns:
39         dataset: float_list
40     """
41     dataset = [list(map(float, data)) for data in dataset]
42     return dataset
```

➤ Kmeans

```

4 class KMeans:
5     def __init__(self, x_train, K):
6         self.x_train = x_train
7         self.K = K
8
9     def euclidean(self, data, center):
10        differ = data - center
11        return np.linalg.norm(differ, axis=1)
12
13    def process(self):
14        center_pos = random.sample(range(0, len(self.x_train)), self.K)
15        center = self.x_train[center_pos] # data[center_pos] have K
16        sigma = np.zeros(len(self.x_train[0]))
17
18        for _ in range(100):
19            arg = np.zeros(len(self.x_train))
20            # 算每個點的距離並給定是哪一群
21            for i in range(len(self.x_train)):
22                arg[i] = np.argmin([self.euclidean(self.x_train[i], center)])
23
24            # 下一次的center更新
25            for c_pos in range(self.K):
26                center[c_pos] = np.mean(self.x_train[arg == c_pos], axis=0)
27
28            for c_pos in range(self.K):
29                sigma[c_pos] = np.sum(self.euclidean(self.x_train[arg == c_pos], center[c_pos])) / len(self.x_train[arg == c_pos])
30
31        return center, sigma

```

隨機挑三個點當中心點，並且初始化 Center 與 sigma

中心點更新，一開始將資料屬於哪個群先記錄在 arg，再把根據 arg 把群體內的點做 mean

算 sigma

➤ Model

```

def train(self, x_train, y_train, window, epoch_setter, loss_setter):
    for i in range(1, self.epochs + 1):
        loss_sum = 0
        for data, label in zip(x_train, y_train):
            # 正向傳遞
            active, self.F = self.predict(data)
            loss_sum += ((label - self.F) ** 2) / 2
            # 倒傳遞更新參數
            new_w = self.w + (self.learning_rate * (label - self.F) * active)
            preprocess_m = self.learning_rate * (label - self.F) * self.w * active * (1 / (self.sigma ** 2))
            data_sub_m = data - self.m
            new_m = self.m + np.array([preprocess_m[i] * data_sub_m[i] for i in range(len(preprocess_m))])
            new_sigma = self.sigma + (self.learning_rate * (label - self.F) * self.w * active * (1 / (self.sigma ** 3)) * (self.euclidean(data, self.F)))
            new_theta = self.theta + (self.learning_rate * (label - self.F))
            self.w = new_w
            self.m = new_m
            self.sigma = new_sigma
            self.theta = new_theta
        if i % 10 == 0:
            print("epoch:", i)
            print(loss_sum)
            print("sum(loss_sum) / len(loss_sum)", loss_sum / len(x_train))
            print("-----")
            epoch_setter.set(str(i))
            loss_setter.set(str(loss_sum / len(x_train)))
            window.update()

```

Forward

Back propagation

每 10 次 epoch 會印一次 loss 在 GUI 上

➤ Simulator

```
9 class Simulator:
10     def __init__(self, car_x, car_y, phi) -> None:
11         self.car_x = car_x
12         self.phi = phi
13         self.b = 3
14         self.coordination = np.array([
15             [-6, -3], [-6, 22], [18, 22], [18, 50], [30, 50], [30, 10], [6, 10], [6, -3], [-6, -3]
16         ])
17
18
19     def rotate(self, theta):
20         theta = np.radians(theta)
21         c, s = np.cos(theta), np.sin(theta)
22         new_point_x = (c * 100) - (s * self.car_y)
23         new_point_y = (s * 100) + (c * self.car_x)
24         return np.array([new_point_x, new_point_y])
25
26     def get_intersection(self, sensor):
27         p1, p2 = Point(self.car_x, self.car_y), Point(sensor[0], sensor[1])
28         l1 = Segment(p1, p2)
29         min_distance = float('inf')
30         for i in range(1, len(self.coordination)):
31             p3, p4 = Point(self.coordination[i - 1][0], self.coordination[i - 1][1]), \
32                 Point(self.coordination[i][0], self.coordination[i][1])
33             l2 = Segment(p3, p4)
34
35             # Point2D
36             intersection = l1.intersection(l2)
37             # 如果有相交
38             if intersection != []:
39                 intersection = intersection[0].evalf()
40                 distance = p1.distance(intersection)
41                 min_distance = min(min_distance, distance)
42
43         return min_distance
```

使用旋轉矩陣，將點定在車的右邊 100，然後用 phi 將車子面相對的位置

取得車與自定義的右邊 100 遠的線使用旋轉矩陣轉成感測器的方向形成的線段與車道邊界相交的點與車位置的距離，挑最短的回傳

```
def start(self, window, canvas, f_plot, feature_len, rbfModel, front_distance, right_distance, left_distance):
```

```
# [前, 右, 左]
four_dimension, six_dimension = [], []
while(self.car_y + 3 < 37):
```

```
# 前方感測器
front_sensor_vec = self.rotate(self.phi)
front_sensor_dis = self.get_intersection(front_sensor_vec)

# 右方感測器
right_sensor_vec = self.rotate(self.phi - 45)
right_sensor_dis = self.get_intersection(right_sensor_vec)

# 左方感測器
left_sensor_vec = self.rotate(self.phi + 45)
left_sensor_dis = self.get_intersection(left_sensor_vec)
```

取得各個感測器抓到與車道的最短距離

```
if(feature_len == 3):
    _, F = rbfModel.predict(np.array([float(front_sensor_dis), float(right_sensor_dis), float(left_sensor_dis)]))
if(feature_len == 5):
    _, F = rbfModel.predict(np.array([float(self.car_x), float(self.car_y), float(front_sensor_dis), float(right_sensor_dis), float(left_sensor_dis)]))
F = F * 80 - 40
```

將感測器抓到的資料丟到 model 進行預測，並將得到的結果正規化回來

```
self.update(F) 更新車的位置與 phi
print(self.car_x, self.car_y, self.phi)
```

```
# 更新前右左距離
front_distance.set(str(front_sensor_dis))
right_distance.set(str(right_sensor_dis))
left_distance.set(str(left_sensor_dis))
window.update()
```

基本的碰撞偵測

```
if front_sensor_dis - 3 < 0 or right_sensor_dis - 3 < 0 or left_sensor_dis - 3 < 0:  
    break
```

將車目前的位置進行畫圖，並將半徑畫出來

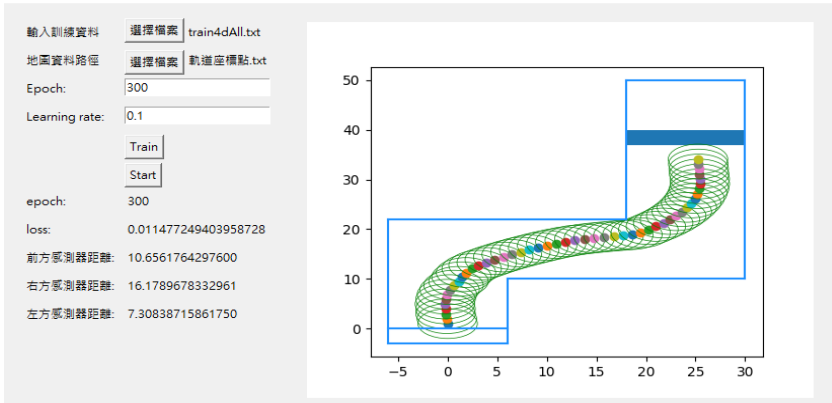
```
84 #動畫畫圖  
85 self.plot(canvas,f_plot)  
86  
87 four_dimension.append([front_sensor_dis,right_sensor_dis,left_sensor_dis,F])  
88 six_dimension.append([self.car_x,self.car_y,front_sensor_dis,right_sensor_dis,left_sensor_dis,F])  
89  
90 return four_dimension,six_dimension  
91  
92 def plot(self,canvas,f_plot):  
93     f_plot.scatter(self.car_x,self.car_y)  
94     circle = patches.Circle((self.car_x,self.car_y), radius=3,linewidth=0.5, fill=False, color="g")  
95     f_plot.add_patch(circle)  
96     canvas.draw()
```

(3) 實驗結果(包含移動軌跡截圖)

Train4dAll.txt


Track4D.txt

神經網路-作業二



```
22.0000000000000 8.48528137423857 8.48528137423858 -6.877838871672125  
21.0939926426109 9.29201086867437 7.76604876868878 -4.465463145168684  
20.2530132392281 25.0691654704510 7.26373714368679 11.087006356142616  
19.0501876320858 8.87574625404333 7.78513987363405 -5.419646410891744  
18.1507977538255 23.7305422503162 7.26723283645272 11.209633267799163  
17.0690742851098 40.7232386997804 7.94244251189077 7.441729136266389  
16.1557040025118 37.8361912675726 8.65341455897487 10.488652868336317  
15.4723449910658 34.6958537190910 10.1827033330733 13.22202761999084  
15.2149667803975 31.9249481889381 13.5120015472074 13.164205151252553  
15.3822956133941 5.52700416320956 14.0311637297385 -22.806349295617885  
12.8525654323397 32.6787303009910 12.0726795509726 13.909152309086217  
12.5807615114606 30.1105744976738 13.0057455211255 13.675496494180024  
12.7150658068294 28.4926800298506 11.2751788053882 14.399736681985082  
13.5272169538935 27.5826581016463 10.0466594730108 14.643397791192584  
15.4402550027892 14.5219307728466 9.27340659139625 -0.9282794776728451  
14.1452548858960 18.4364802358765 8.70430555606019 4.235638951947948  
14.1177694266609 16.8707718184664 8.14085663377995 2.4175243457067666  
13.6958729000118 16.9969901684250 7.61562245828537 2.7932450961068156  
13.4123840541855 16.7616136892137 7.11901510462308 2.6635022328294795  
26.2210200026091 16.5642930155121 6.65154706806835 0.5201677029735663  
25.1720187026397 17.5016677355752 6.19124762773350 1.8888796821177607  
23.8642479828678 17.5822639248489 5.74882721254459 2.843748914972224  
22.4245484058993 17.1256960089817 5.33569906243385 3.5948076463630088  
20.9365670744304 16.3707378580145 4.96322009175058 4.210431054265285  
19.4606918275651 15.4816690409550 4.64202125916116 4.413982121871449  
18.0638204542982 14.6437246899703 4.37661576140161 3.850931157132095  
16.7946758395069 14.0305496300677 4.15664149670038 2.7179941909879943  
15.6446468149515 13.6929994460764 3.96151407008904 1.556377358882763  
14.5733297866381 13.5793688776926 3.77482996153716 0.6754753433987233  
13.5459384839419 13.6178002609743 3.58894487573089 0.13356747416831638  
12.5404853025817 13.7475716967575 33.0265164187241 -21.304034124812635  
13.1365139209446 12.5629116626624 31.0656942320301 -24.49539602690292  
17.2356238390340 10.6076374676952 4.42765462884343 -1.2145175596163682  
16.6454668968970 9.81138118039114 6.17445237055030 -3.559107021377244  
17.0195859927189 9.00885427541924 7.12424196152458 -5.638850667319332  
18.8956965197614 8.26650102067135 7.46146855696663 -6.363991160367604  
22.9967013116415 7.63798559775563 7.60326363244488 -7.131688823895672  
27.8606627059890 7.15270766979080 7.62981121817077 -6.356951268187281  
26.5599742636756 6.78509392942224 7.71931791981825 -7.078162009544215  
25.6108349380090 6.59100857147892 7.74013890678382 -7.548902671491923  
25.0407991861203 6.62161971883813 7.72902525453681 -7.72281847551173  
23.8904157520711 6.94795905545529 7.70613032607020 -7.791692829319345
```


Traind6All.txt

 類神經網路-作業二

輸入訓練資料

選擇檔案

traind6All.txt

地圖資料路徑

選擇檔案

軌道座標點.txt

Epoch:

10

Learning rate:

0.1

Train

Start

epoch:

10

loss:

0.01376033440495231

前方感測器距離:

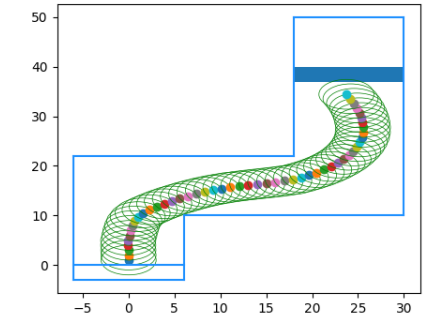
6.69292968360396

右方感測器距離:

17.3627741884807

左方感測器距離:

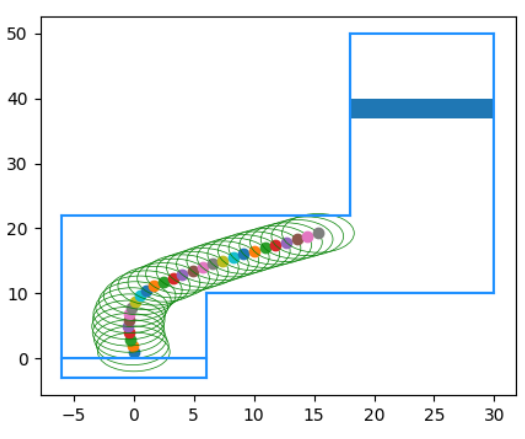
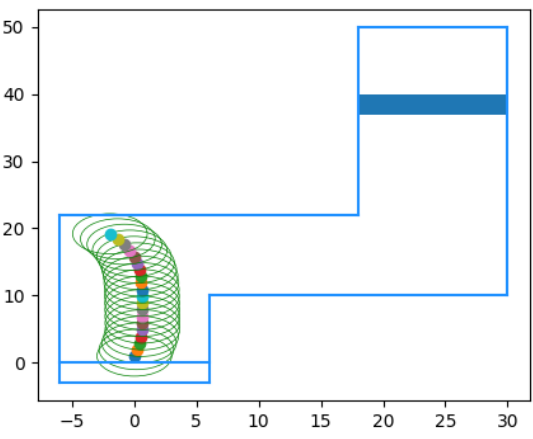
6.39177347727993



Track6D.txt

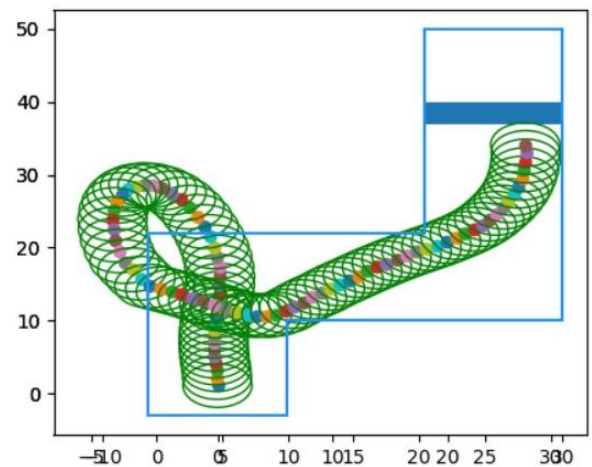
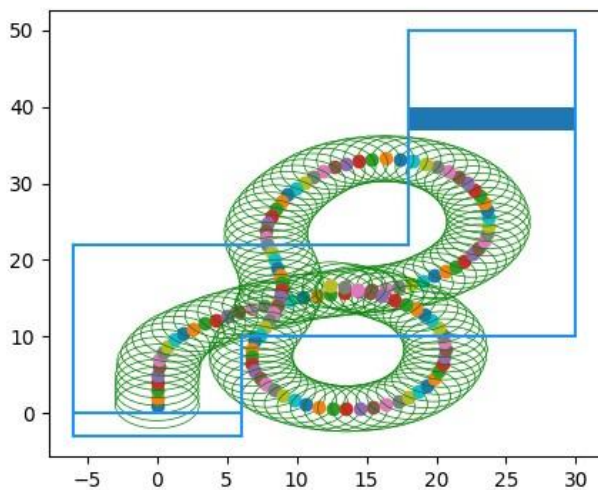
```
-5.898059818321144e-17 0.9998584793474348 22.00000000000000 8.48528137423857 8.48528137423858 -0.9639459065445308
-0.011215354287428484 1.9997834113287012 21.0049637381332 8.60878838508397 8.25358042713015 -0.28270940663635713
-0.025719792542710257 2.9996613266035217 20.0125359359590 8.68159173476245 8.08143026276686 0.3329875575883605
-0.03634915626231569 3.9994962853535574 19.0167980191455 8.69417053586587 7.96374655200086 0.8441898727375658
-0.03714455357758942 4.983508906495768 18.0164684663622 25.1413103410808 7.89560490954112 10.258932952234261
0.08075737072389034 5.97630577372023 17.0611612503828 7.75144528789779 8.74148750009129 1.220259211147794
0.21256538044343168 6.965999265295528 16.0706502698523 38.0580503394562 8.88147051211663 3.208931760009321
0.38025836200066376 7.944502144577164 15.1178669642301 36.8724744241965 9.28157495279705 6.894790322260924
0.6207951869673451 8.887130196648174 14.2891866167662 34.7268585090737 10.3159226956619 13.385671489035914
0.994452116990994 9.759215553045802 13.9083266290627 31.7593649617809 13.3598412084420 18.42039361012109
1.5302027848344912 10.514365340868366 14.4773671046905 29.4372820974075 12.8297531216444 22.196426044234713
2.294988585090638 11.157426066118854 17.0970161322895 5.90748626579082 11.4937828899769 2.274874468730985
3.0752421680990087 11.779002765243822 16.5385919516862 10.1213518986350 10.8456586471607 3.992082749251402
3.8830134032007013 12.363341424251935 16.4428906472938 10.9915488005436 10.2218415336592 4.462535603919462
4.719743384389924 12.904822089035012 16.5899435161113 11.0021937180260 9.65204534986994 4.688650569144906
5.58718553509643 13.401302363332535 29.9330918585610 10.7539660823287 9.14452477673021 1.858176568570062
6.464582050160634 13.87860203058651 28.5775048450817 11.5966351315531 8.65492075893686 2.784760459982742
7.356265415909722 14.326874862250241 27.0521526669896 11.8623216922899 8.19820591664468 3.604383354226769
8.265348303375308 14.737109388883432 25.4364055278054 11.7074354176274 7.78757252177811 4.165384088627626
9.19307280113514 15.102779553831361 23.8259862282580 11.3381289177528 7.43369832830986 4.2964647788123145
10.138313800751996 15.421797052227180 22.3054523524474 10.9368473332742 7.13832003808895 3.958041969085684
11.09797231691189 15.69727801354325 20.9170434763144 10.6149901292970 6.89129509649085 3.225461656401336
12.068066764178722 15.936760240448729 19.6594500000918 10.4180528685327 6.67395879950694 2.267236417600614
13.044689877051786 16.15069477430964 18.5093594946366 10.3590114540296 6.46537624641899 1.200992482156586
14.024420218866034 16.35100550897793 17.4415139660896 10.4435537087505 6.24760633331713 0.11554600936101167
15.004292658166296 16.549973225591007 16.4384762409509 10.6828388481278 6.00877121508149 -0.9271889478864423
15.981560941929512 16.759414218172697 15.4927564182612 11.0990063403491 5.74386969552641 -1.8842926399709938
16.95342587834236 16.99009032538818 14.6052132975356 11.7290868287312 5.45355267996273 -2.7291046290801617
17.88211797926734 17.24191135259709 13.7822525943177 12.6308504715411 33.8773906048420 -15.757808637963857
18.7817815301401 17.673504411277502 14.3942649135078 13.1223738724331 4.75842063236844 -3.7729481681232997
19.65957361425268 18.14303889512655 13.9440591737531 11.9333298549940 9.73904580435698 -5.450191251914632
20.505082965739778 18.666724650205996 13.8764014444738 10.7535055669221 9.72245460999491 -5.98370994127265
21.311608418369055 19.247512282197373 14.1446211029976 9.68441763963390 9.43411336832763 -6.3420965635597994
22.072520603685188 19.88557088038014 14.8871640415203 8.74107262013822 9.17989657877123 -6.769320966152918
22.779826052945733 20.580355605628835 16.4769040245656 7.92799845949690 8.94353270424030 -7.490275559944017
23.42174298834223 21.331252313856155 20.0515795079090 7.25818534973794 8.68419539163774 -8.93021382985656
23.98493132855446 22.1480669646443 29.3291145269753 6.7792017162505 8.34914616193972 -7.182538777433962
24.477077048838083 23.007240731523712 27.9546326150538 6.43746369951310 8.287359327773343 -8.050538280539897
24.886192483484557 23.907149057926993 27.0292857085931 6.30523427898932 8.1787877115917 -8.682867603193145
25.202392782654446 24.842863575150087 26.5982101058631 6.46490447161404 8.05875939005575 -8.997027578837145
25.41855439340537 25.8030588605851 21.7202946201763 7.05323218261149 7.94461021035729 -10.190160471868595
25.520668273649072 26.78890284682178 15.7416575646326 8.62073588495522 7.80659710599141 -7.6419835495337617
```

碰撞偵測



(4) 分析

在 4D 的資料下，目前試下來 Epoch 數量需要比 6D 的多不少，在 4D 的情況下 epoch 拉到 100 – 200 而 learning rate 為 0.1 的情況下都還會撞牆，測試下來 epoch = 300 learning rate = 0.1 效果是最好的，但是 epoch 拉到太高，則會 overfitting，在還沒有碰撞偵測的時候，發生過以下例子，大約是 epoch 1000 以上就會發生



而在 6D 的情況下，因為資料量比較大的關係，只需要不多 epoch 就能不撞牆的到達終點，在 epoch = 10，learning rate = 0.1 的情況下就可以訓練完畢，不過由於 Kmeans 的關係，如果中心點取的不好，那 10 個 epoch 有時候還是發現撞牆的現象，而 epoch 拉到 100 左右，就大致上都能夠不出錯。

(5) 有做的加分

1. RBFN
2. 模擬程式