Import numpy as np --需要用到array

句型配對/TSP

1. 取得資料或自行定義資料

Def F(x):

TARGET\_PHRASE=’sentence’

TARGET\_ASCII = np.fromstring(TARGET\_PHARSE,dtype=np.unit8) –轉換字串成數字

ASCII\_BOUND = [32,126]

N\_CITY = 20 --也是等同DNA size

class TSP(object):

def \_init\_(self,N\_CITY):

self.city\_position = np.random.rand(N\_CITY,2)

env = TSP(N\_CITY)

1. 重要參數

DNA型式ex：[0,0,1,1,0,0,1,0,1,],[1,0,0,0,1,1,0,1,0]

DNA\_SIZE=9=len(TARGET\_PHRASE) --每條DNA的長度：9

POP\_SIZE=2 --population size=共幾條DNA：2

CROSS\_RATE=0.8 --配對機率,TSP時可以較少一點0.1

MUTATE\_RATE=0.003 --變異機率

N\_GENERATIONS=200 --運行次數

X\_BOUND=[0,5] –DNA轉化後之範圍為0~5

1. 五大函式

可以做成一個class

class GA(object):

def \_init\_(self,DNA\_size,cross\_rate,mutation\_rate,pop\_size):

def get\_fittness(pred,line\_x,line\_y) --取得適應度

Return pred +1e-3 – np.min(pred) ---回傳值要>0

Match\_count = (pop == TARGET\_ASCII).sum(axis=1)

-與目標句型比較，對上的字母越多fitness越高

Return match\_count

total\_distance = np.empty((line\_x,shape[0],),dtype = np.float64)

for I,(xs,ys) in enumerate(zip(line\_x,line\_y)):

total\_distance[i] = np.sum(np.sqrt(np.square(np.diff(xs))+ np.square(np.diff(ys))))

fitness = np.exp(DNA\_SIZE \*2/ total\_distance)

return fitness,total\_distance

def translateDNA(pop,city\_position) --翻譯DNA成所求數據型態

Return pop.dot(2 \*\*np.arrange(DATA\_SIZE)[::-1])/(2\*\*DATA\_SIZE-1) \* X\_BOUND

-將二進位之訊息轉為十進位，在濃縮到X\_BOUND的範圍內

Return pop.toString().decode(‘ascii’)

-將2進位以ascii解碼

Line\_x = np.empty\_like(pop,dtype=np.float64)

Line\_y = np.empty\_like(pop,dtype=np.float64)

For I,d in enumerate(pop):

City\_coord = city\_position[d]

line\_x[I,:] = city\_coord[:,0]

line\_y[I,:] = city\_coord[:,1]

return line\_x,line\_y

def select(pop,fitness) --選擇較好適應度的pop

Fitness = get\_fitness()+1e-4

Idx = np.random.choice(np.arrange(POP\_SIZE),size=POP\_SIZE,replace=True,

p=fitness/fitness.sum())

-依據fitness來選pop

Return pop[idx]

def crossover(parent,pop) --交叉配對

If np.random.rand()<CROSS\_RATE:

I\_=np.random.randint(0,POP\_SIZE,size=1)

-從pop中隨機選一個當另一父輩

Cross\_points = np.random.randint(0,2,DNA\_SIZE).astype(np.bool)

-和parent進行隨機crossover的合成位置

Parent[cross\_point] = pop[i\_,crosspoints]

Keep\_city = parent[~cross\_points] --選取從父輩要預留之DNA

Swap\_city = np.setdiff1d(pop[i\_,:],keep\_city) --找到與父輩不同之DNA

Parent[:] = np.concatenate((keep\_city,swap\_city)) --後者補償前者

Return parent

Def mutate(child) --加入差異、隨機換位

For point in range(DNA\_SIZE):

If np.random.rand()<MUTATION\_RATE:

Child[point]=1 if child[point] == 0 else 0

Child[point]=np.random.randint(ASCII\_BOUND)

swap\_point = np.random.randint(0,DNA\_SIZE) –隨機的交換位置

swapA,swapB = child[point],child[swap\_point] –抓取交換前的值

child[point],child[swap\_point] = swapB,swapA –在交換後的位置上賦予值

Return child

1. 主程式運作

初始化每一條DNA並全部存入Pop,可以放在GA.\_init\_裡面

Pop = np.random.randint(0,2,(1,DATA\_SIZE)).repeat(POT\_SIZE,axis=0)

Pop = np.random.randint(\*ASCI\_BOUND,size=(POP\_SIZE,DNA\_SIZE)).astype(np.int8)

Pop = np.vstack([np.random.permutation(DNA\_SIZE) for \_ in range(pop\_size)])

執行迴圈

For \_ in range(N\_GENERATIONS):

F\_value = F(translateDNA(pop))

Fitness = get\_fitness(F\_value)

Best\_DNA = pop[np.argmax(fitness)]

Best\_phrase = translateDNA(best\_DNA)

lx,ly = ga.translateDNA(pop,env.city\_position)

fitness,total\_distance = ga.get\_fitness(lx,ly)

Pop = select(pop,fitness)

Pop\_copy = pop.copy()

For parent in pop:

Child = crossover(parent,pop\_copy)

Child = mutate(child)

Parent[:] = child

best\_idx = np.argmax(fitness)

Np相關使用：

Np.random.rand() – 隨機取值0~1之間的任一數

Np.random.randint(low,high,size,dtype) –隨機取值low(最小數)~high(最大數)之間的整數

Np.random.choice(a,size,replace,p) –將A之內容以P之機率選出

np.setdiff1d(a,b) –找出a中與b不同的地方

np.concatenate((a,b)) –連結a和b