**Bài 22: Thuật toán di truyền - Lập trình AI bằng Python**

Trong bài hôm nay, chúng ta sẽ tìm hiểu chi tiết về thuật toán di truyền trong AI

**1. Thuật toán di truyền là gì ?**

Thuật toán di truyền (GA) là thuật toán tìm kiếm dựa trên các khái niệm về chọn lọc tự nhiên và di truyền. GAs là một tập con của một nhánh tính toán lớn hơn nhiều được gọi là Tính toán tiến hóa.

GA được phát triển bởi John Holland và các sinh viên và đồng nghiệp của ông tại Đại học Michigan, nổi bật nhất là David E. Goldberg. Kể từ đó, nó đã được thử trên các vấn đề tối ưu hóa khác nhau với mức độ thành công cao.

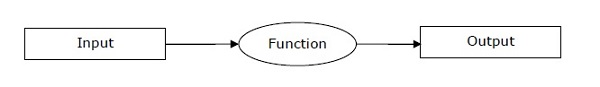
Trong GAs, ta có một nhóm các giải pháp khả thi cho vấn đề đã cho. Những dung dịch này sau đó trải qua quá trình tái tổ hợp và đột biến (giống như trong di truyền tự nhiên), tạo ra những đứa trẻ mới và quá trình này được lặp lại trong nhiều thế hệ khác nhau. Mỗi cá thể (hoặc giải pháp ứng cử viên) được chỉ định một giá trị phù hợp (dựa trên giá trị hàm mục tiêu của nó) và các cá thể lai tạo có cơ hội cao hơn để giao phối và sinh ra các cá thể phù hợp. Điều này phù hợp với Học thuyết của Darwin về sự sống còn của những người khỏe mạnh nhất.

Do đó, nó tiếp tục phát triển các cá nhân hoặc giải pháp tốt hơn qua nhiều thế hệ, cho đến khi đạt được tiêu chí dừng.

Thuật toán di truyền về bản chất là đủ ngẫu nhiên, nhưng chúng hoạt động tốt hơn nhiều so với tìm kiếm cục bộ ngẫu nhiên (nơi ta chỉ thử các giải pháp ngẫu nhiên, theo dõi các giải pháp tốt nhất cho đến nay), vì chúng cũng khai thác thông tin lịch sử.

**2. Cách sử dụng GA cho việc tối ưu hoá vấn đề :**

Tối ưu hóa là một hành động làm cho thiết kế, tình huống, nguồn lực và hệ thống trở nên hiệu quả nhất có thể. Sơ đồ khối sau đây cho thấy quá trình tối ưu hóa:



**a. Các giai đoạn của cơ chế GA cho quá trình tối ưu hóa :**

Sau đây là trình tự các bước của cơ chế GA khi được sử dụng để tối ưu hóa các vấn đề.

1. Tạo ngẫu nhiên quần thể ban đầu.
2. Chọn giải pháp ban đầu với các giá trị phù hợp nhất.
3. Tổng hợp lại các giải pháp đã chọn bằng cách sử dụng các toán tử đột biến và chéo.
4. Đưa một con lai vào quần thể.
5. Bây giờ, nếu điều kiện dừng được đáp ứng, hãy trả lại giải pháp với giá trị thể lực tốt nhất của chúng. Hãy chuyển sang bước 2.

**3. Các thư viện hỗ trợ cần thiết :**

Để giải quyết vấn đề bằng cách sử dụng Thuật toán di truyền trong Python,  tôi sẽ sử dụng một gói mạnh mẽ cho GA có tên là DEAP. Nó là một thư viện của khung tính toán tiến hóa mới để tạo mẫu nhanh và thử nghiệm các ý tưởng. Tôi có thể cài đặt gói này với sự trợ giúp của lệnh sau trên dấu nhắc lệnh:

**pip** install deap

conda:

conda install -c conda-forge deap

**4. Thực hiện các giải pháp sử dụng thuật toán di truyền**

Phần này giải thích cho bạn cách triển khai các giải pháp sử dụng Thuật toán di truyền.

**a. Tạo các mẫu bit**

Ví dụ sau đây cho bạn thấy cách tạo một chuỗi bit chứa 15 chuỗi, dựa trên bài toán One Max.

**import** random

from deap **import** base, creator, tools

Xác định chức năng đánh giá. Đây là bước đầu tiên để tạo ra một thuật toán di truyền.

**def** **eval\_func**(individual):

target\_sum = 15

**return** len(individual) - abs(sum(individual) - target\_sum),

Tạo toolbox với các thông số phù hợp

**def** **create\_toolbox**(num\_bits):

creator.create("FitnessMax", base.Fitness, weights=(1.0,))

creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

toolbox.**register**("attr\_bool", random.randint, 0, 1)

toolbox.**register**("individual", tools.initRepeat, creator.Individual,

toolbox.attr\_bool, num\_bits)

toolbox.**register**("population", tools.initRepeat, list, toolbox.individual)

Đăng kí evaluation operator

toolbox.**register**("evaluate", eval\_func)

Đăng kí crossover operator

toolbox.**register**("mate", tools.cxTwoPoint)

Đăng kí mutation operator :

toolbox.**register**("mutate", tools.mutFlipBit, indpb = 0.05)

Xác định đối tượng để breeding :

toolbox.register("select", tools.selTournament, tournsize = 3)

return toolbox

if \_\_name\_\_ == "\_\_main\_\_":

num\_bits = 45

toolbox = create\_toolbox(num\_bits)

random.seed(7)

population = toolbox.population(n = 500)

probab\_crossing, probab\_mutating = 0.5, 0.2

num\_generations = 10

print('\nEvolution process starts')

Đánh giá toàn bộ population :

fitnesses = list(**map**(toolbox.evaluate, population))

**for** ind, fit in zip(population, fitnesses):

ind.fitness.values = fit

print('\nEvaluated', len(population), 'individuals')

Tạo và lặp lại qua nhiều thế hệ :

**for** g in **range**(num\_generations):

print("\n- Generation", g)

Lựa chọn các cá thể thế hệ tiếp theo

offspring = toolbox.select(population, len(population))

nhân bản các cá thể đã chọn

offspring = list(map(toolbox.clone, offspring))

Áp dụng trao đổi chéo và đột biến trên đời con

**for** **child1**, **child2** **in** **zip**(**offspring**[::2], **offspring**[1::2]):

**if** **random**.random() < **probab\_crossing**:

**toolbox**.mate(**child1**, **child2**)

Xóa giá trị thể chất của trẻ

**del** **child1**.fitness.values

**del** **child2**.fitness.values

áp dụng mauation

**for** **mutant** **in** **offspring**:

**if** **random**.random() < **probab\_mutating**:

**toolbox**.mutate(**mutant**)

**del** **mutant**.fitness.values

Đánh giá những cá nhân có thể trạng không hợp lệ

invalid\_ind = [ind **for** ind in offspring **if** not ind.fitness.valid]

fitnesses = **map**(toolbox.evaluate, invalid\_ind)

**for** ind, fit in zip(invalid\_ind, fitnesses):

ind.fitness.values = fit

print('Evaluated', len(invalid\_ind), 'individuals')

Tiếp theo thay thế population với cá nhân thế hệ tiếp theo

**population[:] = offspring**

In số liệu thống kê cho các thế hệ hiện tại :

fits = [ind.fitness.values[0] **for** ind **in** population]

**length** = **len**(population)

mean = **sum**(fits) / **length**

sum2 = **sum**(x\*x **for** x **in** fits)

**std** = **abs**(sum2 / **length** - mean\*\*2)\*\*0.5

print('Min =', **min**(fits), ', Max =', **max**(fits))

print('Average =', **round**(mean, 2), ', Standard deviation =',

**round**(**std**, 2))

print("\n- Evolution ends")

In ra kết quả cuối cùng :

best\_ind = tools.selBest(population, 1)[0]

print('\nBest individual:\n', best\_ind)

print('\nNumber of ones:', sum(best\_ind))

Following would be the output:

Evolution process starts

Evaluated 500 individuals

- Generation 0

Evaluated 295 individuals

Min = 32.0 , Max = 45.0

Average = 40.29 , Standard deviation = 2.61

- Generation 1

Evaluated 292 individuals

Min = 34.0 , Max = 45.0

Average = 42.35 , Standard deviation = 1.91

- Generation 2

Evaluated 277 individuals

Min = 37.0 , Max = 45.0

Average = 43.39 , Standard deviation = 1.46

… … … …

- Generation 9

Evaluated 299 individuals

Min = 40.0 , Max = 45.0

Average = 44.12 , Standard deviation = 1.11

- Evolution ends

Best individual:

[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,

1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,

1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1]

Number of ones: 15

**5. Symbol Regression Problem**

Đó là một trong những vấn đề được biết đến nhiều nhất trong lập trình di truyền. Tất cả các bài toán hồi quy tượng trưng đều sử dụng phân phối dữ liệu tùy ý và cố gắng khớp dữ liệu chính xác nhất với công thức biểu tượng. Thông thường, một số đo như RMSE (Root Mean Square Error) được sử dụng để đo lường sức khỏe của một cá nhân. Đây là một bài toán hồi quy cổ điển và ở đây chúng ta đang sử dụng phương trình 5x^3-6x^2 + 8x = 1. Ta cần làm theo tất cả các bước như sau trong ví dụ trên, nhưng phần chính sẽ là tạo các tập hợp nguyên thủy vì chúng là các khối xây dựng cho các cá nhân để việc đánh giá có thể bắt đầu. Ở đây chúng ta sẽ sử dụng tập hợp nguyên thủy cổ điển.

Đoạn code sau đây giải thích chi tiết điều này:

**import** **operator**

**import** math

**import** random

**import** numpy as np

from deap **import** algorithms, base, creator, tools, gp

def **division\_operator**(numerator, denominator):

**if** denominator == 0:

**return** 1

**return** numerator / denominator

def eval\_func(individual, points):

func = toolbox.compile(expr=individual)

**return** math.fsum(mse) / len(points),

def create\_toolbox():

pset = gp.PrimitiveSet("MAIN", 1)

pset.addPrimitive(**operator**.add, 2)

pset.addPrimitive(**operator**.sub, 2)

pset.addPrimitive(**operator**.mul, 2)

pset.addPrimitive(division\_operator, 2)

pset.addPrimitive(**operator**.neg, 1)

pset.addPrimitive(math.cos, 1)

pset.addPrimitive(math.sin, 1)

pset.addEphemeralConstant("rand101", lambda: random.randint(-1,1))

pset.renameArguments(ARG0 = 'x')

creator.create("FitnessMin", base.Fitness, weights = (-1.0,))

creator.create("Individual",gp.PrimitiveTree,fitness=creator.FitnessMin)

toolbox = base.Toolbox()

toolbox.**register**("expr", gp.genHalfAndHalf, pset=pset, min\_=1, max\_=2)

toolbox.expr)

toolbox.**register**("population",tools.initRepeat,list, toolbox.individual)

toolbox.**register**("compile", gp.compile, pset = pset)

toolbox.**register**("evaluate", eval\_func, points = [x/10. **for** x in range(-10,10)])

toolbox.**register**("select", tools.selTournament, tournsize = 3)

toolbox.**register**("mate", gp.cxOnePoint)

toolbox.**register**("expr\_mut", gp.genFull, min\_=0, max\_=2)

toolbox.**register**("mutate", gp.mutUniform, expr = toolbox.expr\_mut, pset = pset)

toolbox.decorate("mate", gp.staticLimit(key = **operator**.attrgetter("height"), max\_value = 17))

toolbox.decorate("mutate", gp.staticLimit(key = **operator**.attrgetter("height"), max\_value = 17))

**return** toolbox

**if** \_\_name\_\_ == "\_\_main\_\_":

random.seed(7)

toolbox = create\_toolbox()

population = toolbox.population(n = 450)

hall\_of\_fame = tools.HallOfFame(1)

stats\_fit = tools.Statistics(lambda x: x.fitness.values)

stats\_size = tools.Statistics(len)

mstats = tools.MultiStatistics(fitness=stats\_fit, size = stats\_size)

mstats.**register**("avg", np.mean)

mstats.**register**("std", np.std)

mstats.**register**("min", np.min)

mstats.**register**("max", np.max)

probab\_crossover = 0.4

probab\_mutate = 0.2

number\_gen = 10

population, log = algorithms.eaSimple(population, toolbox,

probab\_crossover, probab\_mutate, number\_gen,

stats = mstats, halloffame = hall\_of\_fame, verbose = True)

Lưu ý rằng tất cả các bước cơ bản giống như được sử dụng trong khi tạo các mẫu bit. Chương trình này sẽ cung cấp cho chúng ta đầu ra là min, max, std (độ lệch chuẩn) sau 10 số thế hệ.

[**BÀI TIẾP THEO: COMPUTER VISION >>**](https://vncoder.vn/bai-hoc/computer-vision-419)