**Deap: python中的遺傳演算法工具箱**

原創2017年10月16日 19:35:33

* 標籤：
* [python](http://so.csdn.net/so/search/s.do?q=python&t=blog) /
* [GA](http://so.csdn.net/so/search/s.do?q=GA&t=blog) /
* [遺傳演算法](http://so.csdn.net/so/search/s.do?q=%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95&t=blog) /
* [deap](http://so.csdn.net/so/search/s.do?q=deap&t=blog)

**Overview 程式概覽**

If you are used to any other evolutionary algorithm framework, you’ll notice we do things differently with DEAP. Instead of limiting you with predefined types, we provide ways of creating the appropriate ones. Instead of providing closed initializers, we enable you to customize them as you wish. Instead of suggesting unfit operators, we explicitly ask you to choose them wisely. Instead of implementing many sealed algorithms, we allow you to write the ones that fit all your needs. This tutorial will present a quick overview of what DEAP is all about along with what every DEAP program is made of.

官方文檔:<http://deap.readthedocs.io/en/master/index.html>   
1. Types : 選擇你要解決的問題類型,確定要求解的問題個數,最大值還是最小值  
2. Initialization : 初始化基因編碼位元數,初始值,等基本資訊  
3. Operators : 操作,設計evaluate函數,在工具箱中註冊參數資訊:交叉(cross over),變異(mutation),保留個體,評價函數(fitness function)

4. Algorithm : 設計main函數,確定參數並運行得到結果

**Types**

he first thing to do is to think of the appropriate type for your problem. Then, instead of looking in the list of available types, DEAP enables you to build your own. This is done with the [creator](http://deap.readthedocs.io/en/master/api/creator.html#module-deap.creator) module. Creating an appropriate type might seem overwhelming but the creator makes it very easy. In fact, this is usually done in a single line. For example, the following creates a FitnessMin class for a minimization problem and an Individual class that is derived from a list with a fitness attribute set to the just created fitness.

# Types

from deap import base, creator

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

# weights :1.0求最大值, -1.0求最小值

# (1.0,-1.0,)求第一個參數的最大值,求第二個參數的最小值

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

That’s it. More on creating types can be found in the [Creating Types](http://deap.readthedocs.io/en/master/tutorials/basic/part1.html) tutorial.

creator用於創建types，上例creator函數創建了目標函數和個體

**Initialization**

Once the types are created you need to fill them with sometimes random values, sometime guessed ones. Again, DEAP provides an easy mechanism to do just that. The [Toolbox](http://deap.readthedocs.io/en/master/api/base.html#deap.base.Toolbox) is a container for tools of all sorts including initializers that can do what is needed of them. The following takes on the last lines of code to create the initializers for individuals containing random floating point numbers and for a population that contains them.

import random

from deap import tools

IND\_SIZE = 10 # 族群數(種群數)

toolbox = base.Toolbox()

toolbox.register("attribute", random.random)

# 呼叫調用randon.random為每一個基因編碼編碼創建 隨機初始值 也就是範圍[0,1]

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

toolbox.attribute, n=IND\_SIZE)

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

創建類型之後需要為其填充資料，Toolbox裡面有許多相關工具

This creates functions to initialize populations from individuals that are themselves initialized with random float numbers. The functions are registered in the toolbox with their default arguments under the given name. For example, it will be possible to call the function toolbox.population() to instantly create a population. More initialization methods are found in the [Creating Types](http://deap.readthedocs.io/en/master/tutorials/basic/part1.html) tutorial and the various [Examples](http://deap.readthedocs.io/en/master/examples/index.html).

**Operators**

Operators are just like initializers, except that some are already implemented in the [tools](http://deap.readthedocs.io/en/master/api/tools.html#module-deap.tools) module. Once you’ve chosen the perfect ones, simply register them in the toolbox. In addition you must create your evaluation function. This is how it is done in DEAP.

# Operators

# difine evaluate function (fittness function)

# Note that a comma is a must

def evaluate(individual):

return sum(individual),

# use tools in deap to creat our application

toolbox.register("mate", tools.cxTwoPoint) # mate:交叉

toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.1) # mutate : 變異

toolbox.register("select", tools.selTournament, tournsize=3) # select : 選擇保留的最佳個體

toolbox.register("evaluate", evaluate) # commit our evaluate

**高斯變異:**

這種變異的方法就是，產生一個服從高斯分佈的亂數，取代原先基因中的實數數值。這個演算法產生的亂數，數學期望當為當前基因的實數數值。一個類比產生的演算法是，產生6個服從U(0,1)的亂數，以他們的數學期望作為高斯分佈亂數的近似。

**mutate方法**

* 這個函數適用於輸入個體的平均值和標準差的高斯突變
* mu: python中基於平均值的高斯變異
* sigma: python中基於標準差的高斯變異
* indpb: 每個屬性的獨立變異概率

**mate : 交叉(交配)**

**select : 選擇保留的最佳個體**

**evaluate : 選擇評價函數,要注意返回值的地方最後面要多加一個逗號**

**Algorithms 計算程式**

也就是設計主程序的地方,按照官網給的模式,我們要早此處設計其他參數,並設計反覆運算和取值的代碼部分,並返回我們所需要的值.

# Algorithms

def main():

# create an initial population of 300 individuals (where

# each individual is a list of integers)

pop = toolbox.population(n=50)

CXPB, MUTPB, NGEN = 0.5, 0.2, 40

'''

# CXPB is the probability with which two individuals

# are crossed (交配率)

#

# MUTPB is the probability for mutating an individual (突變率)

#

# NGEN is the number of generations for which the

# evolution runs

'''

# Evaluate the entire population

fitnesses = map(toolbox.evaluate, pop)

for ind, fit in zip(pop, fitnesses):

ind.fitness.values = fit

print(" Evaluated %i individuals" % len(pop)) # 這時候，pop的長度還是300呢

print("-- Iterative %i times --" % NGEN)

for g in range(NGEN):

if g % 10 == 0:

print("-- Generation %i --" % g)

# Select the next generation individuals

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

# Clone the selected individuals

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

# Change map to list,The documentation on the official website is wrong

# Apply crossover and mutation on the offspring

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

if random.random() < CXPB:

toolbox.mate(child1, child2)

del child1.fitness.values

del child2.fitness.values

for mutant in offspring:

if random.random() < MUTPB:

toolbox.mutate(mutant)

del mutant.fitness.values

# Evaluate the individuals with an invalid fitness

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

# The population is entirely replaced by the offspring

pop[:] = offspring

print("-- End of (successful) evolution --")

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

return best\_ind, best\_ind.fitness.values # return the result:Last individual,The Return of Evaluate function

要注意的地方就是,官網中給出的Overview代碼中有一行代碼是錯誤的,需要把一個資料類型(map)轉換為list.

**輸出結果**

Evaluated 50 individuals

-- Iterative 40 times --

-- Generation 0 --

-- Generation 10 --

-- Generation 20 --

-- Generation 30 --

-- End of (successful) evolution --

best\_ind [-2.402824207878805, -1.5920248739487302, -4.397332290574777, -0.7564815676249151, -3.3478264358788814, -5.900475519316307, -7.739284213710048, -4.469259215914226, 0.35793917907272843, -2.8594709616875256]

best\_ind.fitness.values (-33.10704010746149,)

* best\_ind : 最佳個體
* best\_ind.fitness.values : 最佳個體在經過evaluate之後的輸出

#!usr/bin/env python

# -\*- coding:utf-8 \_\*-

"""

@author:fonttian

@file: Overview.py

@time: 2017/10/15

"""

# Types

from deap import base, creator

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

# weights :1.0求最大值, -1.0求最小值

# (1.0,-1.0,)求第一個參數的最大值,求第二個參數的最小值

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

# Initialization

import random

from deap import tools

IND\_SIZE = 10 # 種群數

toolbox = base.Toolbox()

toolbox.register("attribute", random.random)

# 調用randon.random為每一個基因編碼編碼創建 隨機初始值 也就是範圍[0,1]

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

toolbox.attribute, n=IND\_SIZE)

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

# Operators

# difine evaluate function

# Note that a comma is a must

def evaluate(individual):

return sum(individual),

# use tools in deap to creat our application

toolbox.register("mate", tools.cxTwoPoint) # mate:交叉

toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.1) # mutate : 變異

toolbox.register("select", tools.selTournament, tournsize=3) # select : 選擇保留的最佳個體

toolbox.register("evaluate", evaluate) # commit our evaluate

# Algorithms

def main():

# create an initial population of 300 individuals (where

# each individual is a list of integers)

pop = toolbox.population(n=50)

CXPB, MUTPB, NGEN = 0.5, 0.2, 40

'''

# CXPB is the probability with which two individuals

# are crossed

#

# MUTPB is the probability for mutating an individual

#

# NGEN is the number of generations for which the

# evolution runs

'''

# Evaluate the entire population

fitnesses = map(toolbox.evaluate, pop)

for ind, fit in zip(pop, fitnesses):

ind.fitness.values = fit

print(" Evaluated %i individuals" % len(pop)) # 這時候，pop的長度還是300呢

print("-- Iterative %i times --" % NGEN)

for g in range(NGEN):

if g % 10 == 0:

print("-- Generation %i --" % g)

# Select the next generation individuals

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

# Clone the selected individuals

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

# Change map to list,The documentation on the official website is wrong

# Apply crossover and mutation on the offspring

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

if random.random() < CXPB:

toolbox.mate(child1, child2)

del child1.fitness.values

del child2.fitness.values

for mutant in offspring:

if random.random() < MUTPB:

toolbox.mutate(mutant)

del mutant.fitness.values

# Evaluate the individuals with an invalid fitness

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

# The population is entirely replaced by the offspring

pop[:] = offspring

print("-- End of (successful) evolution --")

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

return best\_ind, best\_ind.fitness.values # return the result:Last individual,The Return of Evaluate function

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

# t1 = time.clock()

best\_ind, best\_ind.fitness.values = main()

# print(pop, best\_ind, best\_ind.fitness.values)

# print("pop",pop)

print("best\_ind",best\_ind)

print("best\_ind.fitness.values",best\_ind.fitness.values)

# t2 = time.clock()

# print(t2-t1)

版權聲明：如果您覺得本文不錯,麻煩滑鼠右轉點下贊,謝謝.如果有什麼問題,請直接留言.轉載只需標明原文網址即可. https://blog.csdn.net/FontThrone/article/details/78253230

* 本文已收錄於以下專欄：
* [劍指數據科學](https://blog.csdn.net/column/details/16266.html)