Генетичните алгоритми с Python

В темата ще бъдат разгледани следните основни въпроси:

- Основна идея на генетичните алгоритми
- Основни термини и стъпки на генетичните алгоритми
  - Основни термини в генетичните алгоритми
  - Основни стъпки на генетичните алгоритми
- Реализиране на генетичен алгоритъм с Python
  - Инициализиране на генетичния алгоритъм
  - Изчисляване на фитнес функцията
  - Кръстосване
  - Мутация
  - Селекция
  - Прекратяване на алгоритъма

1. Основна идея на генетичните алгоритми

Генетичният алгоритъм GA (Genetic Algorithm) е алгоритъм вдъхновен от естествената природата на еволюцията, който широко се използва за решаване на проблеми с оптимизацията. Принадлежи към клона на апроксимационните алгоритми, защото не гарантира винаги намирането на точното оптимално решение; въпреки това може да намери почти оптимално решение за ограничено време.

Генетичните алгоритми симулират процес подобен на естествените системи за еволюция. Според създадената от Чарлз Дарвин теория на еволюцията, биологичните същества при естествения подбор се развиват според принципа на „оцеляване на най-силните“. По подобен начин търсенето в GA е проектирано така, че да насърчава теорията за „оцеляването на най-силните“.

Оптимизацията е процес на намиране на най-доброто решение на определен проблем. Например задачи за намиране на най-краткия маршрут до местоназначение, планиране на болничния персонал по възможно най-ефективния начин или планиране на дейностите за деня, за да се използва най-добре наличното време. За да се решат проблемите с оптимизацията в реалния свят, проблемите се формулират като математически функции и оптимизацията се занимава с минимизиране/максимизиране на изхода на тази функция, за да се намери най-доброто решение. След като проблемът е формулиран математически, се използва алгоритъм за оптимизация като GA, за да се намери най-доброто решение (оптимално решение) на този проблем.

Идеята за генетичните алгоритми е предложена от Джон Холанд и неговите студенти в Мичиганския университет през 1960 г и за първи път

описана в книга от него през 1975 г. [?]. Генетичният алгоритъм е еволюционен алгоритъм, вдъхновен от процеса на естествен подбор. Естественият подбор, според Дарвин, е механизъм, чрез който популациите на различни видове се адаптират и еволюират. Най-силните индивиди оцеляват и възпроизвеждат подобно потомство, докато слабите индивиди се елиминират с течение на времето. Предложеният от Джон Холанд GA генерира на случаен принцип множество от кандидат-решения за определен оптимизационен проблем. По-добрите решения получават по-голям шанс да произведат подобни решения чрез извършване на кръстосване. Едно ново поколение ще наследи повече от по-добрите решения, което ще доведе до оцеляване на характеристиките на по-добрите решения.

2. Основни термини и стъпки на генетичните алгоритми

Накратко ще бъдат разгледани основните термини, с които борави един генетичен алгоритъм, и стъпките му, наречени генетични оператори, чрез които той имитира процеса на естествения подбор.

2.1. Основни термини в генетичните алгоритми

Основните термини, с които работи един генетичен алгоритъм, са:

Хромозома. Хромозомата е потенциално решение на решавания проблем. Решението може да не е непременно оптималното решение. Хромозомата може да се разглежда като масив от двоични/целочислени променливи, където всяка променлива може да се разглежда като ген, а стойността на променливата се нарича алел.

Популация. Множеството от решения, участващи в процеса на оптимизация, се нарича популация. За разлика от алгоритмите, базирани на траектория, GA е алгоритъм, базиран на популацията, в който няколко

решения взаимодействат помежду си, за да намерят глобалния оптимален резултат.

Целева функция. Целевата функция често се нарича и фитнес функция. Всеки път, когато се решава оптимизационен проблем, той първо се формулира като математическа функция, която оценява качеството на кандидата за решение. Обикновено се предава определено решение на тази функция и функцията връща годността на това решение. След като бъде намерено най-подходящото (оптимално) решение, процесът се спира.

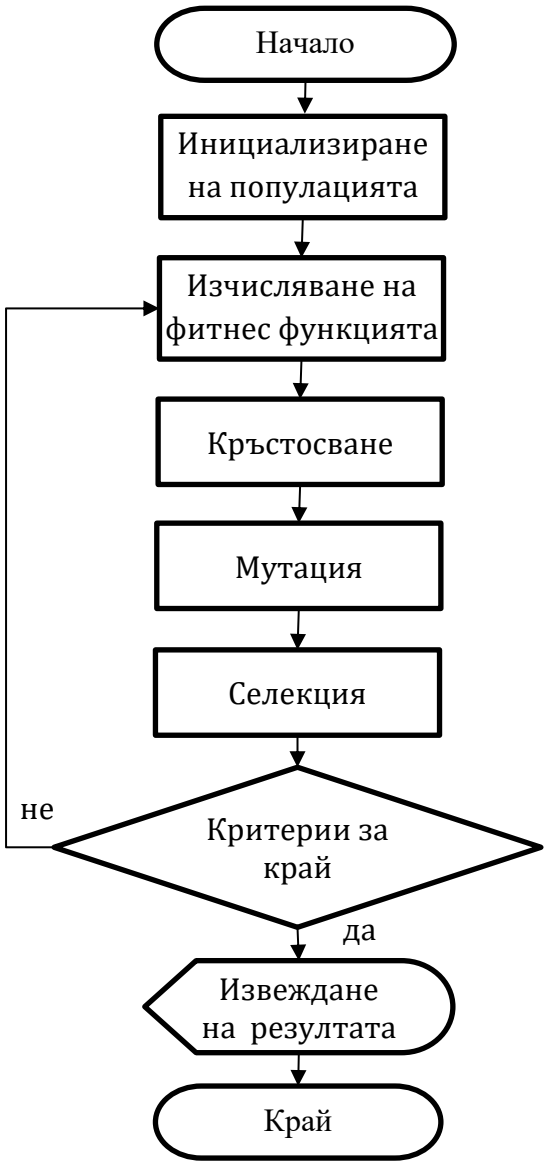
Основните термини, използвани в генетичния алгоритъм, са визуализирани на фиг. 1.

Фиг. 1. Основни термини, използвани в генетичните алгоритми

2.2. Основни стъпки в генетичните алгоритми

Генетичният алгоритъм имитира процеса на естествен подбор, който включва няколко стъпки, наречени генетични оператори. Първо се генерира популация от произволни решения (хромозоми). След това се изчислява годността на всеки кандидат за решение чрез изчисление на фитнес

функцията. Следва последователно изпълнение на генетичните оператори кръстосване, мутация и селекция, докато се достигнат критериите за край на генетичния алгоритъм, както е показано на фиг. 2.



Фиг. 2. Основни стъпки на генетичния алгоритъм

3. Реализиране на генетичен алгоритъм с Python

Реализацията на генетичен алгоритъм с Python ще бъде представена с пример за максимизиране на проста математическа функция с две променливи:

$$f(x_1, x_2) = x_1^2 + x_2^2.$$

Подробно описание на примера и неговата реализация можете да намерите в статията „Въведение в генетичните алгоритми с Python” на адрес https://algodaily.com/lessons/introduction-to-genetic-algorithms-in-python .

3.1. Инициализиране на генетичния алгоритъм

При инициализацията на генетичния алгоритъм се използват начални стойности на променливи и библиотеката numpy, защото тя позволява случайно инициализиране на масиви (фиг. 3.а). Пример за случайно генерирана популация е показана на фиг. 3.б, където редовете са 6-те хромозоми, а колоните двете гени. Границите, в които ще се максимизира функцията са зададени от двете променливи lb и ub, което определя затворения интервал [-3, 3].

```
import numpy

# Parameter initialization
genes = 2
chromosomes = 6
mattingPoolSize = 4
offspringSize = chromosomes - mattingPoolSize
lb = -3
ub = 3
populationSize = (chromosomes, genes)
generations = 3

#Population initialization
population = numpy.random.uniform(lb, ub, populationSize)
```

а)

```
population
[[ 2.20349714 -1.63247799]
 [ 1.4170337  -0.16585167]
 [-1.81699671  2.70148615]
 [-2.69779288  1.06947395]
 [-2.77573091 -1.36449618]
 [ 0.46067933 -0.34050889]]
```

б)

Фиг. 3. Инициализиране на генетичния алгоритъм

а. Код на Python **б.** Изход от програмата

3.2. Изчисляване на фитнес функцията

Изчисляването на фитнес функцията $f(x_1, x_2) = x_1^2 + x_2^2$ на цялата популация се извършва с кода от фиг. 4. Например за **хромозома 1** = [2.20349714 -1.63247799], фитнес функцията е $f(x_1, x_2) = 2.20349714^2 + (-1.63247799)^2 = 7.5203840338226197$.

```
print(("Generation:", generation+1))
fitness = numpy.sum(population*population, axis=1)
print("\npopulation")
print(population)
print("\nfitness calcuation")
print(fitness)
```

а)

```
('Generation:', 1)

population
[[ 2.20349714 -1.63247799]
 [ 1.4170337  -0.16585167]
 [-1.81699671  2.70148615]
 [-2.69779288  1.06947395]
 [-2.77573091 -1.36449618]
 [ 0.46067933 -0.34050889]]

fitness calcuation
[ 7.52038404  2.03549128 10.59950445  8.42186094  9.56653194
0.32817175]
```

б)

Фиг. 4. Изчисляване на фитнес функцията

а. Код на Python **б.** Изход от програмата

3.3. Кръстосване

За да се извърши кръстосване, се изисква първо да се изгради мутиращият масив, който ще съдържа най-добрите решения за участие в

оператора кръстосване. Броят на родителите ще се определя от променливата mattingPoolSize, която в примера има стойност 4.

Създава се празен двумерен масив за родителите с метода numpy.empty() на фиг. 5. След това в цикъл се определя индекса на хромозомата с максимална фитнес функция. В резултат се определят родителите с максимални фитнес функции. Това са четирите хромозоми с най-големи фитнес функции: [10.59950445 9.56653194 8.42186094 7.52038404]. Те са изведени на фиг. 5.б.

```
    # Following statement will create an empty two dimensional
    array to store parents
        parents = numpy.empty((mattingPoolSize,
    population.shape[1]))

        # A loop to extract one parent in each iteration
        for p in range(mattingPoolSize):
            # Finding index of fittest chromosome in the
    population
            fittestIndex = numpy.where(fitness ==
    numpy.max(fitness))
            # Extracting index of fittest chromosome
            fittestIndex = fittestIndex[0][0]
            # Copying fittest chromosome into parents array
            parents[p, :] = population[fittestIndex, :]
            # Changing fitness of fittest chromosome to avoid
    reselection of that chromosome
            fitness[fittestIndex] = -1
        print("\nParents:")
        print(parents)
```

а)

```
Parents:
[[-1.81699671  2.70148615]
 [-2.77573091 -1.36449618]
 [-2.69779288  1.06947395]
 [ 2.20349714 -1.63247799]]
```

б)

Фиг. 5. Определяне на родителите с максимални фитнес функции

а. Код на Python **б.** Изход от програмата

Операторът за кръстосване използва предварително определения брой потомци от родители, избрани с кода от фиг. 5.а, като използва **кръстосване с една точка**. Празен двумерен масив за съхраняване на потомство се създава с метода numpy.empty((offspringSize, population.shape[1])). В кода (фиг. 6.а) всяко k-то потомство има първата половина от родителя по индекс (k mod mattingPoolSize) от родителския масив и втората половина от родителя по индекс ((k+1) mod mattingPoolSize). Трябва да се отбележи, че има много други начини за избор на родители, като например многоточково кръстосване, равно вероятно кръстосване и др. Размерът на масива на потомството е offspringSize = chromosomes – mattingPoolSize = 6 – 4 = 2.

```
    # Following statement will create an empty two dimensional
    array to store offspring
        offspring = numpy.empty((offspringSize,
    population.shape[1]))
        for k in range(offspringSize):
            #Determining the crossover point
            crossoverPoint = numpy.random.randint(0,genes)

            # Index of the first parent.
            parent1Index = k%parents.shape[0]

            # Index of the second.
            parent2Index = (k+1)%parents.shape[0]

            # Extracting first half of the offspring
            offspring[k, 0: crossoverPoint] =
    parents[parent1Index, 0: crossoverPoint]

            # Extracting second half of the offspring
            offspring[k, crossoverPoint:] =
    parents[parent2Index, crossoverPoint:]
        print("\nOffspring after crossover:")
        print(offspring)
```

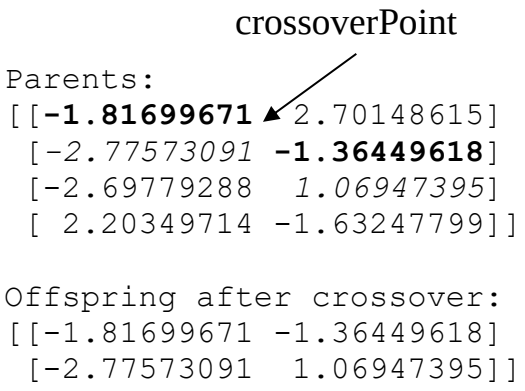
а)

```
Offspring after crossover:
[[-1.81699671 -1.36449618]
 [-2.77573091  1.06947395]]
```

б)

Фиг. 6. Определяне на родителите с максимални фитнес функции
а. Код на Python **б.** Изход от програмата

За пояснение работата на тази част от алгоритъма ще бъдат показани резултатите от работата му за първата генерация (фиг. 7). Точката за мутиране crossoverPoint се определя произволно с метода numpy.random.randint(0,genes). В случая от фиг. 7 crossoverPoint = 1. Потомството с индекс 0 има първа половина от родителя по индекс 0 = 0 mod 4 от родителския масив и втората половина от родителя по индекс 1 = 1 mod 4 (изборът е показан с удебелен шрифт). Потомството с индекс 1 има първа половина от родителския масив с индекс 1 и втора половина от родителя с индекс 2 (изборът е показан с курсив).



Фиг. 7. Определяне на родителите с максимални фитнес функции

3.4. Мутация

Прилага се мутация с произволна инициализация, защото алелите са непрекъснати стойности. Генът на всяко потомство ще бъде избран на случаен принцип и инициализиран с произволна стойност (фиг. 8).

```
    # Implementation of random initialization mutation
        for index in range(offspring.shape[0]):
            randomIndex = numpy.random.randint(1,genes)
            randomValue = numpy.random.uniform(lb, ub, 1)
            offspring [index, randomIndex] = offspring [index,
    randomIndex] + randomValue
        print("\n Offspring after Mutation")
        print(offspring)
```

а)

```
Offspring after Mutation
[[-1.81699671  1.55833979]
 [-2.77573091 -0.17909233]]
```

б)

Фиг. 8. Мутация с произволна инициализация

а. Код на Python **б.** Изход от програмата

3.5. Селекция

В етапа на селекция (естествен подбор) родителите и новосъздаденото потомство колективно ще изградят следващото поколение. С други думи, слабите хромозоми от предишното поколение ще бъдат заменени с новото потомство. Кодът на етапа селекция е показан на фиг. 9.

```
        population[0:parents.shape[0], :] = parents
            population[parents.shape[0]:, :] = offspring
            print("\nNew Population for next generation:")
            print(population)
```

а)

```
New Population for next generation:
[[-1.81699671  2.70148615]
 [-2.77573091 -1.36449618]
 [-2.69779288  1.06947395]
 [ 2.20349714 -1.63247799]
 [-1.81699671  1.55833979]
 [-2.77573091 -0.17909233]]
```

б)

Фиг. 9. Селекция на новата популация за следващата генерация

а. Код на Python **б.** Изход от програмата

3.6. Прекратяване на алгоритъма

Критериите за прекратяване на генетичния алгоритъм могат да бъдат определени по няколко начина:

1. Изпълняват се предварително определен брой цикли.
2. Получава се предварително зададена стойност на фитнес функцията.
3. Няма подобрение в резултатите на фитнес функцията за фиксиран брой повторения.

В конкретната реализация е използван най-простия начин, като се изпълняват точно определен брой цикли, зададени с променливата generation (виж фиг. 3.а).

Извеждането на резултата в края на генетичния алгоритъм е показано на фиг. 10.

```
fitness = numpy.sum(population*population, axis=1)
fittestIndex = numpy.where(fitness == numpy.max(fitness))
# Extracting index of fittest chromosome
fittestIndex = fittestIndex[0][0]
# Getting Best chromosome
fittestInd = population[fittestIndex, :]
bestFitness = fitness[fittestIndex]
print("\nBest Individual:")
print(fittestInd)
print("\nBest Individual's Fitness:")
print(bestFitness)
```

а)

```
Best Individual:
[-1.81699671 -4.11834511]

Best Individual's Fitness:
20.2622434865039
```

б)

Фиг. 8. Извеждане на резултата от генетичния алгоритъм

а. Код на Python **б.** Изход от програмата