Testing an implementation of the GA on 10 famous test functions

Mh. Samadi

***Abstract*— In this essay we'll test a python GA library called pygad on 10 famous test functions implemented with numpy. We'll take a closer look at its moving parts and we'll try and solve a world-class problem using GA.**

I. INTRODUCTION

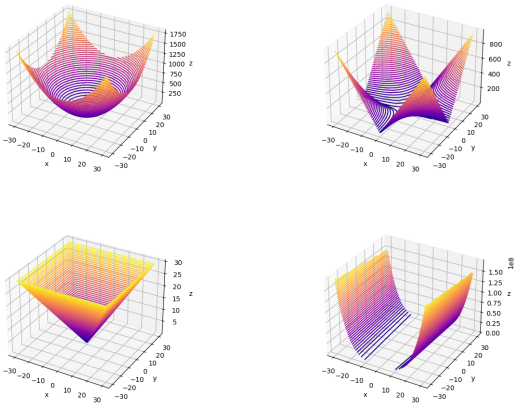
In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.
In this essay we'll use PyGAD. PyGAD is an open-source Python library for building the genetic algorithm and optimizing machine learning algorithms. It works with Keras and PyTorch.
The assignment had 4 different parts. The first part asks for the average fitness of the final solution of the algorithm over 10 instances of it ran for 3 different dimensionalities of 10 different test functions. The second part asks for different values for the moving parts of the algorithm such as, population size and crossover probability. The third demands a solution to a real world problem using the GA algorithm. Finally the forth part brings up a question about the effects of running multiple instances of the algorithm in parallel and how it effects the performance overall.

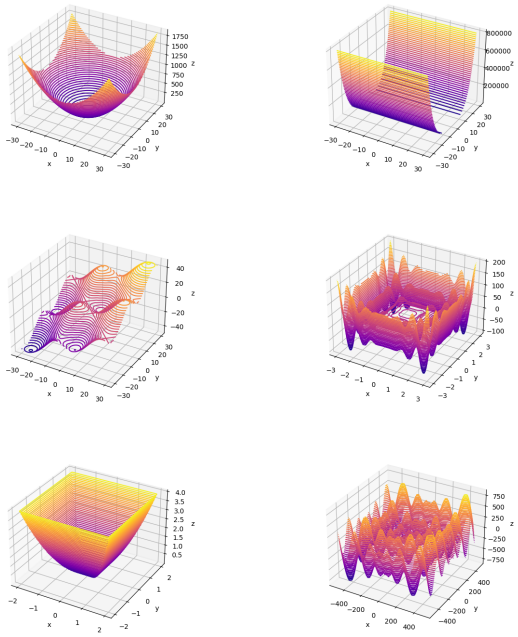
II. IMPLEMENTATION

For the first part the implementation is made of 2 different loops. One over 10 different test functions, another over 3 different dimensionalities (10, 30 and 50). The test functions used are thoroughly explained in their own dedicated subsection (2.A). The 2 mentioned loops are wrapped with a final loop for running each of the mentioned instances of the algorithm for 10 times and averaging over the varying results to get a better estimate of the algorithm's utility.
For the second part we'll use a strange idea. We'll run another GA algorithm (we'll call it the tuner instance) to determine the moving parts of the main GA algorithm trying to maximize a 2D f5 test function. For this to work, the fitness function of the tuner instance runs an instance of GA algorithm with the parameters specified by the tuner instance for 500 generations and yields the final fitness of it as the fitness for the tuner instance.
As for the third part demanding to solve a real-world problem. Let's say we want to re-create a picture using GA to put the algorithm to practice on a high dimensioned problem.
As for the final part, we'll just run different instances of the algorithm on different parts of a test function.

A. Test Function

The test functions used in this essay are shown in the following section:





from left to right, top to bottom we have f1, f2, f4, f5, f6, f7, f8, rastrigin, schwefel and schwefel problem test functions.

III. Results

Results for the first part of the assignment are:

-	10D	30D	50D
f1	-1.8e-5	-0.17	-1.13
f2	-0.006	-1.64	-499
f4	-0.008	-0.272	-0.5
f5	-194	-27	-2965
f6	-2.2	-0.16	-1.21
f7	-1.74	-12.42	-27.46
f8	529	1568	2702
rastrigin	83e4	11e5	17e5
schwefel	-1.2e-5	-0.06	-0.28
schwefelP	519	1616	2737

As for the second part. A the values found for the GA algorithm trying to maximize an upside down f5 by another GA was: population size of 16, crossover probability of 0.751 and mutation probability of 0.342. The final fitness achieved on the f5 with these parameters was -1.83e-5 which is fairly close to the results achieved by the algorithm in the first part but it's found by another GA algorithms and not by a researcher.
For the third part, the reference image and the image guessed by GA is showed here:

picture on the right is the final guess of the left picture by GA

For the last part, the algorithm was ran for multiple domains over the f5 function. If ran in parallel on a multi-core cpu, the final result will be much faster and the resolution of the search will increase if we use the same parameters for each instance.