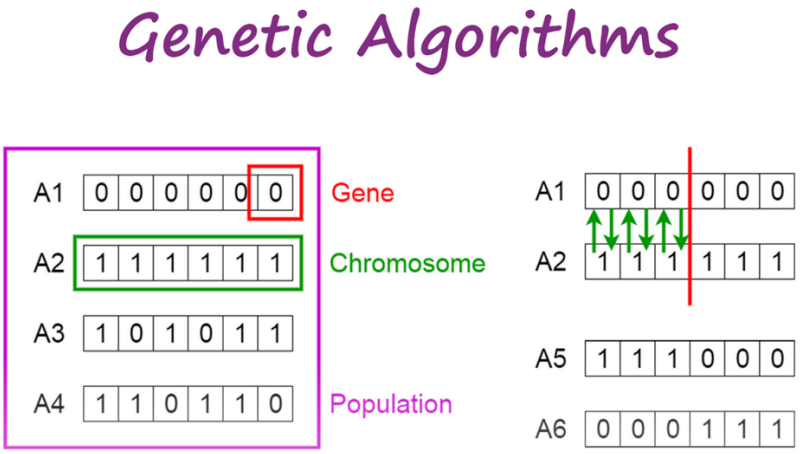
CS 451 Introduction to Artificial Intelligence

Homework 2

Mert Erkol S017789

Introduction

A genetic algorithm is a search heuristic inspired by Charles Darwin's theory of natural evolution. The algorithm reflects that the process of natural selection and among the population, the fittest individuals will survive and create the next generation. Genetic algorithm has some inspired and similar functions to biology like selection, crossover, and mutation functions. The Genetic Algorithm (GA) is commonly used for optimization problems like solving sudoku puzzles, decision trees, and hyperparameter optimization.



The GA is reported in 1963 by Nils Aal Barricelli. But although his work on artificial evolution become popular with the help of Ingo Rechenberg and Hans-Paul Schwefel in the early 1970s. When John Holland used evolutionary programming in his work The GA algorithm become much more popular in the 1970s. Research in GAs was still theoretical until the mid-1980s when the GA conference was held in Pittsburgh, Pennsylvania.

Methodology

The GA methodology is easy to understand and execute because it operates the same way our chromosomes cross their genes and create their genetic variation. The GA implementation is consisting of 5 main steps are

- Initial Population
- Fitness Calculation
- Selection
- Crossover
- Mutation

The initial population is the first step of our algorithm firstly we just create random chromosomes that have genes on them we can decide the number of chromosomes it is up to us and this is our initial population. Our initial population can have constraints like the number of genes, chromosomes, or the number of values that our gene can have. We are free to play at this step. After we create our population, we need to define our fitness function.

The fitness function is totally up to us for example what are trying to maximize or minimize is simply an evolution function like f1-score in machine learning. After each generation, we create, we need to calculate the fitness score of our chromosomes.

When we created our population and calculated the fitness scores of each chromosome, we need to select or matting pool. A Matting pool is simply the count of parents who need to be chosen. If our goal is to maximize the fitness score, we are going to select k best highest fitness scores. And these chromosomes will generate our next generation. Also, we can decide to keep the parents for the next generation which is called Elitism.

After we decide on our parents the next step is crossing over our parents to create more genetic variation and this can be done by swapping our genes from some point if we have only one point which is called single-point crossover. This will create our first offspring child we can also create 2 different offspring but, in my implementation, I will create only one offspring from 2 parents.

After our parents' offspring, we need to mutate the random genes with random numbers to create a more genetic variation to reach a minimum or maximum fitness more quickly. This can be done with flip-mutation, inverse mutation, or a random number generated from some interval.

After all these steps we eventually created our next generation, we can do these steps multiple times to reach a better fitness score and we can also define our limitations like stop after some number of generations created or some fitness score has been reached.

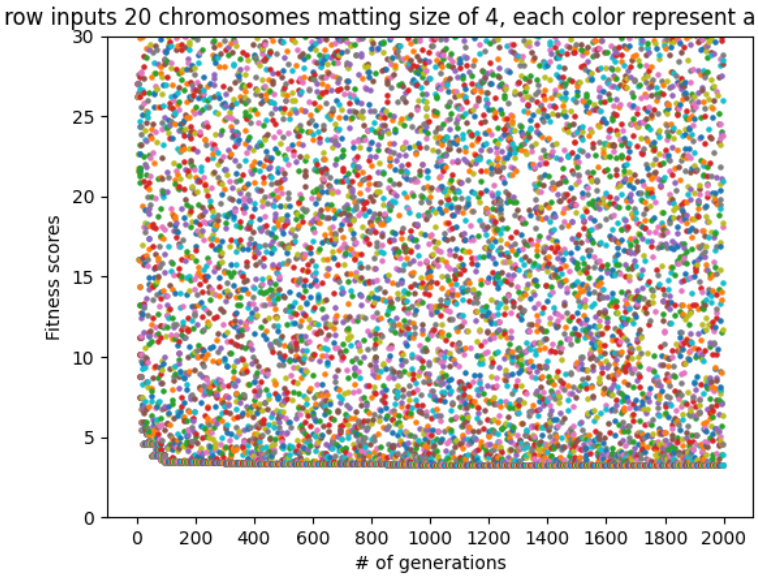
Implementation Details

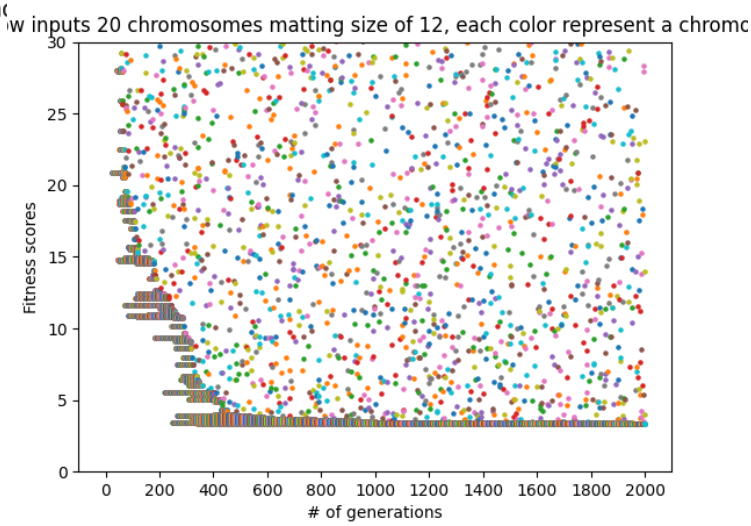
As also described in the homework document our goal is to minimize the fitness score based on our weights (genes). We have a data_generator.py file to generate some X and Y data for us. In the homework description, the formula to calculate our prediction and fitness score has given all we need to do is transform them into meaningful code.

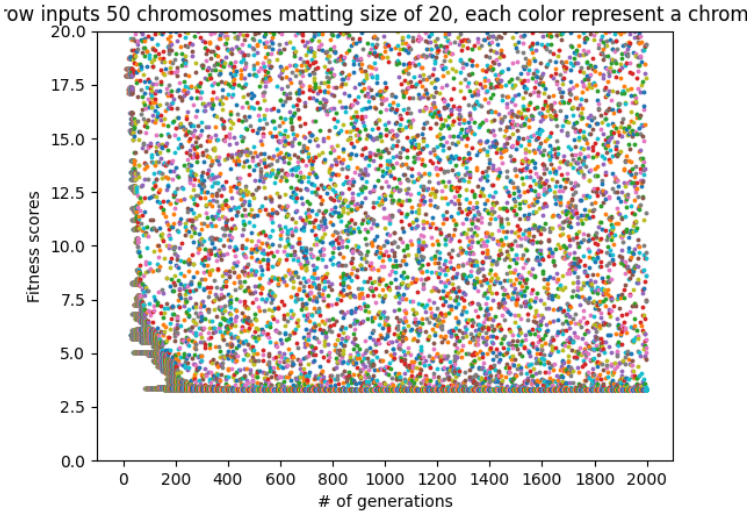
To perform this prediction calculation operation, I firstly needed to transform my matrix to put them into a dot product formula. Let's say I have several 20 populations and 12 genes to calculate the y_predict of each chromosome first I transformed each row to (3,2) and (2,3) matrices with the help of split from NumPy with this reshaped matrix I can now use np. dot to calculate my prediction for my input matrix of (100,3). I have used sample_data.csv for my calculations and I didn't change my data_generator.py file. After I calculated the fitness scores and put them to a NumPy array size of my population. I selected parents based on the mattingSize number. If you put this number to 12 let's, say you can only generate the chromosomes-12 size of offspring since I am using elitism. I select the parents based on their fitness function and who has minimum fitness score. Then the crossover begins, to create my offspring, kth offspring will have the first half from kth parent and the second half from the k+1th parent with that I only produce one child from 2 parents and also the crossover point selected randomly to generate more variations. Also, to generate more variations I implemented a mutation. This mutation goes on a loop in offspring and selects a random index from a range of genes and changes that gene to a random value between the given intervals. After all these operations we successfully generated our next generation. If we have a mattingSize of 12, we will have 12 parents selected with elitism and 8 offspring produced from these parents. We will do this operation until we generate exactly 2000 generations.

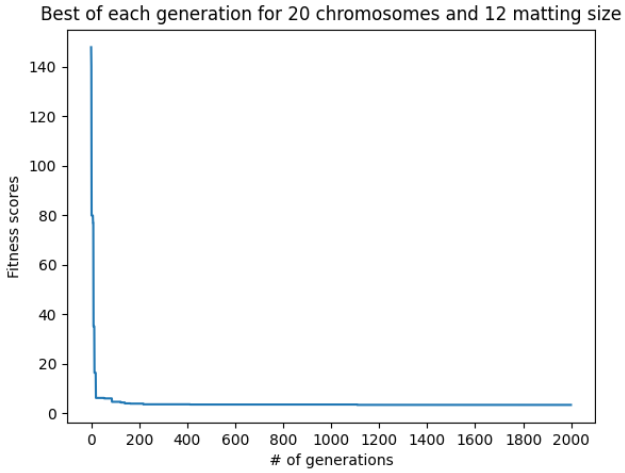
Results

Here below you can see some runs with their respected graphs all these runs made MacBook Pro 2019 i9 10th gen we can see that increasing the matting size affected reaching minimum fitness









Conclusion

We can see that we can use GA for optimization problems and yes there are limitations for GA now like they do not scale well in time but in the future. I believe we are going to see more evolving algorithms to solve long-time-taking problems and tunings in deep learning. In near future, I believe this scalability is also going to increase well in time.

References

- https://en.wikipedia.org/wiki/Genetic_algorithm