Ben Lovelace

Professor Michael

DASE 4570

5/7/2020

Genetic Algorithm Report

Genetic algorithms (GA) belong to the larger class of evolutionary algorithms and can be used where other techniques such as gradient descent falls short. GA are a metaheuristic inspired by natural selection in nature and the concept of Darwin’s theory of evolution. GA starts with an initial population of random members, from here the individual members are receive a fitness rating as a result of their inputs into a fitness function. Next, only a few of the highest fitness members are kept for mating. The most popular form of mating includes two parents producing two children. In the case of binary genetic algorithms, the children are produced via the exchange of binary code around selected crossover point(s). After children are produced, mutations are introduced into the population, excluding the moist elite member. These mutations take form of switching the value of a bit from a 1 to a 0 or visa versa. Mutation points are randomly selected among the population. The introduction of mutations allows the algorithm to explore more of the search space and not converge too quickly. The number of generations iterated depends on whether the acceptable solution is discovered, or the number of iterations is met. For the assignment to convert the MATLAB code to r, I ended up scrapping the idea of using a library because I felt it was too complicated for the simple binary genetic algorithm that we had to create. Instead of using a premade library of functions, I ended up writing the functions myself. I actually learned more about the structure of a genetic algorithm and how to use r better this way. My binary genetic algorithm relies on the inputs or population members to be 5 bits long. I decided on 5 bits by restricting my search space to 32 or 2^5. The fitness function I chose to optimize was f(x) = abs(x-5) + sin(x-5) + 5. This function’s minimum is at x=5, where y = 5. The first function I wrote was the mate function. This function receives the top 4 performing population member’s binary strings and determined the crossover point to be everything before the 4th bit. From here I crossed parent 1 and 2 then parent 3 and 4 to create an entire new population matrix. The next function I created was the compareAndRank function. This receives an entire population matrix and ranks them from most fit to least fit. The function does this by decoding the binary value and entering it into the fitness function. It then appends the decimal output of the fitness function to the binary string of the member. After all members are evaluated, the matrix is then ordered from least to greatest based upon the output from the fitness function at the end of the binary string. Once the matrix is ordered, the fitness function output is dropped and the ordered binary matrix is output. The last function I needed to create was the mutate function. In this function I determine the number of mutations by following the formula in the book. From here I produce random column and row values to be mutated (excluding the fittest member). Then the selected values are removed from the matrix and replaced with the opposite binary symbol. The mutated matrix is then returned. At the bottom of the program, an initial population of random binary values is produced. This initial population will be used in the for loop beneath it. The for loop will iterate through a set iteration number of 200. Inside this for loop, the population is passed into the compareAndRank function, then the mate function, then to the mutate function and finally through the compare and rank function. This is repeated 200 times and the minimum x coordinate will be the first member of the population. The program might need to be run a few times in order for it to find the minimum.