

Experiment 5

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate “survival of the fittest” among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Five phases are considered in a genetic algorithm.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

Initial Population

The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve.

An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution).

In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.

Fitness Function

The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Selection

The idea of the selection phase is to select the fittest individuals and let them pass their genes to the next generation.

Two pairs of individuals (parents) are selected based on their fitness scores. Individuals with high fitness have more chances to be selected for reproduction.

Crossover

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes. For example, consider the crossover point to be 3 as shown below.

Mutation

In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

Mutation: Before and After

Mutation occurs to maintain diversity within the population and prevent premature convergence.

Termination

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

ALGORITHM:

```
Initialize a random population of individuals
Compute fitness of each individual
WHILE NOT finished DO
  BEGIN /* produce new generation */
    FOR population_size DO
      BEGIN /* reproductive cycle */
        Select two individuals from old generation, recombine the two
        individuals to give two offspring
        Make a mutation for selected individuals by altering a random bit
        in a string
      END
    END
  END
  Create a new generation (new populations)
END
IF population has converged THEN finished :=
TRUE
END
```

Why use Genetic Algorithms

- They are Robust
- Provide optimisation over large space state.

- Unlike traditional AI, they do not break on slight change in input or presence of noise •
- Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

CODE:

```
from numpy.random import randint from
numpy.random import rand import math

def binary_to_decimal(bin):
    decimal=0 for i in
    range(len(bin)):
        decimal+=bin[i]*pow(2, 4-i)
    return decimal

def decimal_to_binary(dec): binaryVal=[]
    while(dec>0):
        binaryVal.append(dec%2)
        dec=math.floor(dec/2) for _ in
        range(5-len(binaryVal)):
            binaryVal.append(0)
        binaryVal=binaryVal[::-1] return
        binaryVal

def crossover(parent1,parent2,r_cross): child1,child2 =
    parent1.copy(), parent2.copy() r = rand() point = 0
    if r > r_cross: point = randint(1,len(parent1)-2)
    child1 = parent1[:point] + parent2[point:] child2 =
    parent2[:point] + parent1[point:]
    return child1,child2,point
```

```

def mutation(chromosome,r_mut):
    for i in range(len(chromosome)):
        if rand()<r_mut:
            chromosome[i] = 1 - chromosome[i]
    return chromosome

def fitness_function(x): return
    pow(x,2)

def genetic_algorithm(iterations, population_size, r_cross, r_mut):
    input = [randint(0, 32) for _ in range(population_size)] pop = [decimal_to_binary(i) for i in
    input] for generation in range(iterations): print(f'\nITERATION : {generation+1}',end='\n\n')
    decimal = [binary_to_decimal(i) for i in pop] fitness_score = [fitness_function(i) for i in
    decimal] f_by_sum = [fitness_score[i]/sum(fitness_score) for i in range(population_size)]
    exp_cnt = [fitness_score[i]/(sum(fitness_score)/population_size) for i in
    range(population_size)] act_cnt = [round(exp_cnt[i]) for i in range(population_size)]
    print('SELECTION\n\nInitial \tDecimal Value\tFitness Score\t\tFi/Sum\t\tExpected
\tActual ') for i in range(population_size):

    print(pop[i],'\t',decimal[i],'\t\t',fitness_score[i],'\t\t',round(f_by_sum[i],2),'\t\t',round(exp_cnt[i],2),'\t\t',a
    ct_cnt[i]) print('Sum : ',sum(fitness_score)) print('Average : ',sum(fitness_score)/population_size)
    print('Maximum : ',max(fitness_score),end='\n') max_count = max(act_cnt) min_count = min(act_cnt)
    max_count_index = 0 for i in range(population_size):
        if max_count == act_cnt[i]:
            maxIndex=i break
    for i in range(population_size): if min_count ==
        act_cnt[i]: pop[i] =
            pop[max_count_index]
    crossover_children = list() crossover_point = list() for i in
    range(0,population_size,2): child1, child2, point_of_crossover =
    crossover(pop[i],pop[i+1],r_cross) crossover_children.append(child1)

```

```

        crossover_children.append(child2)
    crossover_point.append(point_of_crossover)
    crossover_point.append(point_of_crossover) print("\nCROSS
OVER\n\nPopulation\t\tMate\t Crossover Point\t Crossover
Population") for i in range(population_size): if
    (i+1)%2 == 1:
        mate = i+2
    else: mate = i
    print(pop[i],'\t',mate,'\t',crossover_point[i],'\t\t\t',crossover_children[i])
    mutation_children = list() for i in range(population_size):
    child = crossover_children[i]
    mutation_children.append(mutation(child,r_mut))
    new_population = list() new_fitness_score = list() for i in mutation_children:
    new_population.append(binary_to_decimal(i)) for i in new_population:
    new_fitness_score.append(fitness_function(i)) print("\nMUTATION\n\nMutation
population\t New Population\t Fitness") for i in range(population_size):
    print(mutation_children[i],'\t',new_population[i],'\t\t\t',new_fitness_score[i])
    print('Sum : ',sum(new_fitness_score)) print('Maximum : ',max(new_fitness_score))
    pop = mutation_children
print("-----")

def main(): iterations = 3 population_size = 4 r_cross = 0.5 r_mut = 0.05
    genetic_algorithm(iterations,population_size,r_cross,r_mut)

if __name__ == '__main__':
    main()

```

OUTPUT:

ITERATION : 1

SELECTION

Initial	Decimal Value	Fitness Score	Fi/Sum	Expected	Actual
[0, 1, 0, 0, 1]	9	81	0.11	0.43	0
[1, 0, 1, 0, 0]	20	400	0.53	2.12	2
[0, 0, 1, 1, 1]	7	49	0.06	0.26	0
[0, 1, 1, 1, 1]	15	225	0.3	1.19	1
Sum : 755					
Average : 188.75					
Maximum : 400					

CROSS OVER

Population	Mate	Crossover Point	Crossover Population
[0, 1, 0, 0, 1]	2	0	[0, 1, 0, 0, 1]
[1, 0, 1, 0, 0]	1	0	[1, 0, 1, 0, 0]
[0, 1, 0, 0, 1]	4	2	[0, 1, 1, 1, 1]
[0, 1, 1, 1, 1]	3	2	[0, 1, 0, 0, 1]

MUTATION

Mutation population	New Population	Fitness
[0, 1, 0, 0, 1]	9	81
[1, 0, 1, 0, 0]	20	400
[0, 1, 1, 1, 1]	15	225
[0, 1, 0, 0, 1]	9	81
Sum : 787		
Maximum : 400		

ITERATION : 2

SELECTION

Initial	Decimal Value	Fitness Score	Fi/Sum	Expected	Actual
[0, 1, 0, 0, 1]	9	81	0.1	0.41	0
[1, 0, 1, 0, 0]	20	400	0.51	2.03	2
[0, 1, 1, 1, 1]	15	225	0.29	1.14	1
[0, 1, 0, 0, 1]	9	81	0.1	0.41	0
Sum : 787					
Average : 196.75					
Maximum : 400					

CROSS OVER

Population	Mate	Crossover Point	Crossover Population
------------	------	-----------------	----------------------

[0, 1, 0, 0, 1]	2	0	[0, 1, 0, 0, 1]
[1, 0, 1, 0, 0]	1	0	[1, 0, 1, 0, 0]
[0, 1, 1, 1, 1]	4	2	[0, 1, 0, 0, 1]
[0, 1, 0, 0, 1]	3	2	[0, 1, 1, 1, 1]

MUTATION

Mutation population	New Population	Fitness
[0, 1, 0, 0, 1]	9	81
[1, 0, 1, 0, 0]	20	400
[0, 1, 1, 0, 1]	13	169
[0, 1, 1, 1, 1]	15	225
Sum :	875	
Maximum :	400	

ITERATION : 3

SELECTION

Initial	Decimal Value	Fitness Score	Fi/Sum	Expected	Actual
[0, 1, 0, 0, 1]	9	81	0.09	0.37	0
[1, 0, 1, 0, 0]	20	400	0.46	1.83	2
[0, 1, 1, 0, 1]	13	169	0.19	0.77	1
[0, 1, 1, 1, 1]	15	225	0.26	1.03	1
Sum :	875				
Average :	218.75				
Maximum :	400				

CROSS OVER

Population	Mate	Crossover Point	Crossover Population
[0, 1, 0, 0, 1]	2	1	[0, 0, 1, 0, 0]
[1, 0, 1, 0, 0]	1	1	[1, 1, 0, 0, 1]
[0, 1, 1, 0, 1]	4	0	[0, 1, 1, 0, 1]
[0, 1, 1, 1, 1]	3	0	[0, 1, 1, 1, 1]

MUTATION

Mutation population	New Population	Fitness
[0, 0, 1, 0, 0]	4	16
[1, 1, 0, 0, 1]	25	625
[0, 1, 1, 0, 1]	13	169
[0, 1, 1, 0, 1]	13	169
Sum :	979	
Maximum :	625	

CONCLUSION:

We learnt about the Genetic Algorithm, its workings and its uses and also implemented it in a python program. We also learnt about other terms associated with genetic algorithm such as crossover, mutation, fitness score, etc.