Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Assignment

Course No: CSE 4203

Course Title: Neural Networks and Fuzzy Systems

Submitted By

Saifur Rahman

Roll: 1703018

Sec: A

Dept. of CSE, RUET

Submitted To

Dr. Md. Rabiul Islam

Professor

Dept. of CSE, RUET

Date of Submission: 10th June, 2023

Contents

1. [Assignment Name 1](#_Toc137265039)
2. [Objectives 1](#_Toc137265040)
3. [Problem Definition 1](#_Toc137265041)
4. [Methodology 1](#_Toc137265042)
5. [Implementation 2](#_Toc137265043)
6. [Results & Performance Analysis 4](#_Toc137265044)
7. [Conclusion & Observation 6](#_Toc137265045)

# Assignment Name

Implementation of Genetic Algorithm to find out the bit sets that can optimize the search of minimum number.

# Objectives

* To find the combination of bits in a chromosome that results in the lowest possible decimal value.
* To iteratively evolve the population through selection, crossover, and mutation operations to improve the fitness.
* To identify the minimum number and the corresponding best chromosome after a certain number of generations.

# Problem Definition

This problem involves a genetic algorithm to find out the bit sets that can optimize the search of minimum number. The fitness function or objective function must be defined to find out the minimum number. The basic genetic operators such as reproduction or selection, crossover and mutation should be used to optimize the searching criteria. The chromosome (string) with minimum 20 genes (bits) and population 30. The number of generations may be used for the breaking condition. Followings might be reported:

* Initial random strings and fitness function
* Output of reproduction, crossover and mutation for each generation
* Calculate the efficiency of each generation

# Methodology

1. Initialize the genetic algorithm parameters: chromosome length, population size, and number of generations.
2. Generate an initial random population consisting of binary strings.
3. Print the initial population along with their fitness values.
4. Set the minimum number as infinity and the best chromosome as None.
5. Repeat the following steps for the specified number of generations:
   1. Create a new population by selecting parent chromosomes, performing crossover, and introducing mutations.
   2. Update the population with the new population by including children.
   3. Display the current generation and the fitness values of the chromosomes.
   4. Check if any chromosome has a lower fitness value (minimum number) than the current minimum number. If yes, update the minimum number and store the corresponding best chromosome.
   5. Calculate and display the efficiency of the current population.
6. Print the minimum number and the best chromosome found by the genetic algorithm.
7. End the algorithm.

# Implementation

1. *# -\*- coding: utf-8 -\*-*
2. """Genetic Algorithm CT #4 Assignment
4. Automatically generated by Colaboratory.
6. Original file is located at
7. https://colab.research.google.com/drive/1h\_eF-h3rhCTOwweVum9JkV6lltf7\_jVz
9. """
11. **import** random
13. *# Genetic Algorithm parameters*
14. chromosome\_length = 20
15. **population\_size = 30**
16. num\_generations = 10
18. """Fitness Function"""
20. **def fitness\_function(chromosome):**
21. decimal\_value = int(chromosome, 2) *# Convert binary string to decimal*
22. **return** decimal\_value
24. """Generate Initial Population"""
26. **def** generate\_population():
27. population = []
28. **for** \_ **in** range(population\_size):
29. chromosome = ''.join(random.choice(['0', '1']) **for** \_ **in** range(chromosome\_length)) *# generate random chromosome*
30. **population.append(chromosome)**
31. **return** population
32. """Roulette Wheel Selection"""
34. **def selection(population):**
35. total\_fitness = sum(fitness\_function(chromosome) **for** chromosome **in** population)
36. probabilities = [fitness\_function(chromosome) / total\_fitness **for** chromosome **in** population]
37. selected = random.choices(population, probabilities, k=2) *# select 2 chromosome randomly by using probabilities*
38. **return** selected[0], selected[1]
40. """Single-Point Crossover"""
42. **def** crossover(parent1, parent2):
43. crossover\_point = random.randint(1, chromosome\_length - 1)
44. **child1 = parent1[:crossover\_point] + parent2[crossover\_point:]**
45. child2 = parent2[:crossover\_point] + parent1[crossover\_point:]
46. **return** child1, child2
48. """Bit Flip Mutation"""
50. **def** mutation(chromosome, mutation\_rate):
51. mutated\_chromosome = ''
52. **for** bit **in** chromosome:
53. **if** random.random() < mutation\_rate:
54. **mutated\_chromosome += '0' if bit == '1' else '1' *# inverted***
55. **else**:
56. mutated\_chromosome += bit *# unchanged*
57. **return** mutated\_chromosome
59. **"""Each Generation Efficiency Calculation"""**
61. **def** calculate\_efficiency(population):
62. **return** min(fitness\_function(chromosome) **for** chromosome **in** population)
64. **"""Genetic Algorithm"""**
66. **def** genetic\_algorithm():
68. *# generate random population*
69. **population = generate\_population()**
70. **print**("Initial Population:")
72. *# print population with fitness values*
73. **for** chromosome **in** population:
74. **print(chromosome, fitness\_function(chromosome))**
75. **print**()
77. minimum\_number = float('inf')
78. best\_chromosome = None
80. **for** generation **in** range(num\_generations):
81. new\_population = []
83. **for** \_ **in** range(population\_size // 2): *# no of parent pair*
84. **parent1, parent2 = selection(population) *# selection***
85. child1, child2 = crossover(parent1, parent2) *# crossover*
86. child1 = mutation(child1, mutation\_rate=0.01) *# mutation*
87. child2 = mutation(child2, mutation\_rate=0.01) *# mutation*
88. new\_population.extend([child1, child2])
90. population = new\_population
92. **print**("Generation", generation + 1)
93. **for** chromosome **in** population:
94. **fitness = fitness\_function(chromosome)**
95. **print**(chromosome, fitness) *# fitness value*
96. **if** fitness < minimum\_number:
97. minimum\_number = fitness
98. best\_chromosome = chromosome
100. **print**("Efficiency:", calculate\_efficiency(population)) *# efficiency of current generation*
101. **print**()
103. **print**("Minimum Number:", minimum\_number)
104. **print("Best Chromosome:", best\_chromosome)**
106. genetic\_algorithm()

# Results & Performance Analysis

Initial Population:

01001011110010101101 310445

10101011000100010010 700690

11111100001110110010 1033138

01101101001110110100 447412

11000111111111001000 819144

11110111000110010101 1012117

11000000010011110000 787696

11111110101111110110 1043446

11100111110110111010 949690

00010011001001101110 78446

10000011001111000001 537537

01001000100110100100 297380

00000001100001111100 6268

11111110100011111000 1042680

11001111000111110111 848375

11011001011110111110 890814

10010101110000111111 613439

10011110011010110110 648886

00010100001011100011 82659

11110010011010111101 992957

11100111011101100101 948069

00010011110110111101 81341

10101000000001001010 688202

10100111101010011010 686746

11011111010010111001 914617

10100110110010000101 683141

10001011111110101100 573356

00110011000111011110 209374

01010110011111101001 354281

01110100111010110100 478900

Generation 1

11100111110010101101 949421

01001010011101100101 304997

10000011111111101001 540649

01010110011110101100 354220

01111110101101110101 519029

11110111000110010110 1012118

11001111100011111000 850168

11111110000111110111 1040887

11000000010011110000 787696

10101000000001001010 688202

10001110101111110110 584694

11111011111110101100 1032108

10010101110000111010 613434

10100111101010011111 686751

11011001000111110111 889335

11001111011110111110 849854

11111111000010111001 1044665

11011110001111110110 910326

11011111010010011010 914586

10100111101010111001 686777

11111101001110110100 1037236

01100111000110010101 422293

01010110011110101100 354220

10001011111111101001 573417

11100111010111110111 947703

11001111001101100101 848741

11000000011111101001 788457

01010110000011110000 352496

11100111000111110111 946679

11001111110110111010 851386

Efficiency: 304997

Generation 2

11111101001110101100 1037228

01010110011110110100 354228

11000000010011110000 787696

11000000010011110000 787696

11000111000010010110 815254

11111111011110111110 1046462

01111011111110101100 507820

11111110101101110101 1043317

11000000011111110000 788464

11000000010011111001 787705

11001111001101100101 848741

01001010011001100101 304741

11110111000110010110 1012118

10001110101111110110 584694

11110111110110111010 1015226

11001111000110010110 848278

11000000000011111000 786680

11001111111111101001 851945

11100111010111110111 947703

11100111000111110111 946679

01100111011101100001 423777

01001010010110010101 304533

10001110101111110111 584695

11111110000111110110 1040886

10101110000111110111 713207

11110111101010011111 1014431

11000000010010101101 787629

11100111111111101001 950249

11111111000010111001 1044665

10001011111111101001 573417

Efficiency: 304533

Generation 3

11110111101110101100 1014700

11111101001010011111 1036959

01111011111110100001 507809

01100111010101101100 423276

11111111011010011111 1046175

11110111101110111110 1014718

11110111000110010110 1012118

00000000000011011000 216

11001111000110010111 848279

11000000010011111000 787704

11111110100110010110 1042838

11001111001101110101 848757

01111011111110101001 507817

10001011111111101100 573420

10101110000111110110 713206

11000111000010010111 815255

11001111001101010110 848726

11001111000110100101 848293

11000001110110111010 794042

11110110010011111001 1008889

11100010101101110101 928629

11111000010011110000 1017072

11001001010011111001 824569

11000111111111101001 819177

11001110010011110000 845040

01000001111111101001 270313

11000000000111110111 786935

11100111010011111000 947448

11100011000111111001 930297

10001011101111100111 572391

Efficiency: 216

Generation 4

01000001111111101111 270319

11111101001010011001 1036953

11110111101110110111 1014711

11000111000010011110 815262

11111111011010111111 1046207

11110111101110101100 1014700

11110110010011111001 1008889

11111101001010011111 1036959

01110111101110101100 490412

11111011111110101001 1032105

11101110100110010110 977302

11111111000110010111 1044887

11000000010010010110 787606

11110111000111111000 1012216

11110000000111100111 983527

11000111101110111110 818110

11111100100110010110 1034646

11111111001010011111 1045151

11110110010011110101 1008885

11100010101101111001 928633

11100010101101110101 928629

11100010101101110101 928629

01000011111111101100 278508

10001001111111101001 565225

11000000000111110110 786934

11000001110110111011 794043

11111101000110100101 1036709

11001111101011011111 850655

11000111000011110111 815351

11000000000110010111 786839

Efficiency: 270319

Generation 5

11100010101101110101 928629

11111101000110100101 1036709

11110110010011110111 1008887

11110111101110111001 1014713

11000001010110111011 791995

11000001110110011011 794011

11100010101101111001 928633

11000000010010010110 787606

11111101101110101001 1039273

11111011110110010110 1031574

11000111001101111001 815993

11100010100010011110 927902

11110000000111101110 983534

11101110100110010111 977303

11100010101101110001 928625

11110110010011111101 1008893

11001111101011011111 850655

11000011110110111011 802235

11111111101110101100 1047468

11110100100110010110 1001878

11111110100110010111 1042839

11111111000110010110 1044886

11100000000110010111 917911

11011011111110101001 901033

11111101000110100100 1036708

11101110100110010111 977303

11111101000110010110 1036694

11101110100110100101 977317

11111111000111111000 1044984

11110111000111111000 1012216

Efficiency: 787606

Minimum Number: 216

Best Chromosome: 00000000000011011000

# 

# Conclusion & Observation

* The selection process uses the Roulette Wheel Selection method.
* Single-Point Crossover technique is used.
* Bit Flip Mutation is applied.
* Efficiency of each generation is calculated based on the minimum fitness value in the population.
* The algorithm aims to optimize the search for the minimum number by evolving the population over generations.
* The results obtained from the algorithm provide valuable insights into the efficiency and effectiveness of the genetic algorithm.