LAB - 05 **(27-11-24)**

Grey Wolf Optimizer (GWO):
The Grey Wolf Optimizer (GWO) algorithm is a swarm intelligence algorithm inspired by the social hierarchy and hunting behavior of grey wolves. It mimics the leadership structure of alpha, beta, delta, and omega wolves and their collaborative hunting strategies. The GWO algorithm uses these social hierarchies to model the optimization process, where the alpha wolves guide the search process while beta and delta wolves assist in refining the search direction. This algorithm is effective for continuous optimization problems and has applications in engineering, data analysis, and machine learning.

Implementation Steps:
1.Define the Problem: Create a mathematical function to optimize.
2. Initialize Parameters: Set the number of wolves and the number of iterations.
3. Initialize Population: Generate an initial population of wolves with random positions.
4. Evaluate Fitness: Evaluate the fitness of each wolf based on the optimization function.
5.Update Positions: Update the positions of the wolves based on the positions of alpha, beta, and delta wolves.
6. Iterate: Repeat the evaluation and position updating process for a fixed number of iterations or until convergence criteria are met. 7.Output the Best Solution: Track and output the best solution found during the iterations

ALGORITHM/LOGIC-

PURPOSE:

Cuckoo search algorithm is used to find an optimal ~~search~~ or near optimal solution to complex problems by observing how certain species of cuckoo birds use brood parasitism by laying their eggs in the nest of other host birds, exploiting the concept of Lévy flight to explore and exploit the solution space.

APPLICATION:

The cuckoo search algorithm is used in training of ANN (artificial neural networks), robots, wireless sensor networks, classification, clustering, design optimization, images etc.

- In ANN, cuckoo search was used to test limitations of ANN.
- It was used in path planning problems for an autonomous mobile robot.
- ~~Used for finding optimal clustering object into clusters using multiple datasets.~~
- power problem in power distribution system. (minimize power loss & improve voltage)
- distribution of networ reconfiguration

ALGORITHM:

objective function $f(x)$, $x = (x_1, \ldots, x_d)^T$
Generate initial population of n host nests x_i
while ($t <$ MaxGeneration) or (stop criterion)

Rule 1 {
Get a cuckoo randomly
Generate a solution by Lévy flights
Evaluate its solution quality or objective value f_i
}

Rule 2 {
choose a nest among n (say, j) randomly
if ($f_i < f_j$), Replace j by the new solution
end
}

Rule 3 {
A fraction (p_α) of worse nests are abandoned
New nests / solutions are built / generated
keep best solutions (or nests with quality solutions)
Rank the solutions and find the current best
Update $t \leftarrow t+1$
}

end while

IMPLEMENTATION:

There are three basic rules followed for implementation of cuckoo search algorithm:

Rule 1:

Each cuckoo lays one egg at a time and dumps it in a randomly chosen nest. (each egg represents one solution)

Rule 2:

Best nest with high quality eggs will

be carried over to next generation.
→ High quality eggs → best solution
Rule 3:
The number of available host nests are fixed, egg laid by cuckoo bird being discovered has probability $pa \in (0, 1)$
→ number of host nest = fixed population
→ host bird discovering egg = worst solution

MATHEMATICAL MODEL:
Lèvy flight:

- $x_i^{t+1} = x_i^t + \alpha \otimes L(\lambda)$
  new solution → x_i^{t+1}; existing solution → x_i^t; step size → α; entry-wise multiplication → $\otimes$; levy exponent → $L(\lambda)$
- $Levy(\lambda) \approx g^{-\lambda} \quad 1 < \lambda \leq 3$
- step size →

$$s = \frac{U}{|V|^{1/\lambda}}$$

where $U \approx N(0, \sigma_u^2)$; $V \approx N(0, \sigma_v^2)$

$$\sigma_u^2 = \left[\frac{\Gamma(1+\lambda) . \sin(\pi\lambda/2)}{\Gamma(1+\lambda/2) \times \lambda \times 2^{\lambda-1/2}}\right]^{1/\lambda} ; \sigma_v^2 = 1$$

Local Random Walk: (markov chain whose next loc depends on current)
$x_i^{t+1} = x_i^t + H(Pa - \varepsilon) \otimes (x_j^t - x_k^t)$
- $H(u)$ is Heavside function
- Pa is switching parameter
- ε is random number $\in$ uniform dist
- x_j and x_k are two diff solutions selected randomly

OUTPUT:
Generation 1: Best fitness: 81.4347
Generation 2: Best fitness: 70.8612
⋮
Generation 50: Best fitness : 23.7909

Optimal solution found:
Best solution [2.3888 1.4004 3.8218]
Best fitness: 23.7909

27/11/24

INPUT-

```
import numpy as np
def objective_function(x):
    return np.sum(x**2)

class GreyWolfOptimization:
    def __init__(self, n_agents, n_variables, max_iter, lower_bound, upper_bound):
        self.n_agents = n_agents
        self.n_variables = n_variables
        self.max_iter = max_iter
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound

        self.position = np.random.uniform(lower_bound, upper_bound, (n_agents, n_variables))

        self.fitness = np.zeros(n_agents)

        self.alpha_pos = np.zeros(n_variables)
        self.beta_pos = np.zeros(n_variables)
        self.delta_pos = np.zeros(n_variables)

        self.alpha_score = float("inf")
        self.beta_score = float("inf")
        self.delta_score = float("inf")

    def update_positions(self, a):
        for i in range(self.n_agents):
            r1 = np.random.random(self.n_variables)
            r2 = np.random.random(self.n_variables)

            A1 = 2 * a * r1 - a
            C1 = 2 * r2
            D_alpha = np.abs(C1 * self.alpha_pos - self.position[i, :])
            X1 = self.alpha_pos - A1 * D_alpha

            r1 = np.random.random(self.n_variables)
            r2 = np.random.random(self.n_variables)

            A2 = 2 * a * r1 - a
            C2 = 2 * r2
```

```
            D_beta = np.abs(C2 * self.beta_pos - self.position[i, :])
            X2 = self.beta_pos - A2 * D_beta

            r1 = np.random.random(self.n_variables)
            r2 = np.random.random(self.n_variables)

            A3 = 2 * a * r1 - a
            C3 = 2 * r2
            D_delta = np.abs(C3 * self.delta_pos - self.position[i, :])
            X3 = self.delta_pos - A3 * D_delta

            self.position[i, :] = (X1 + X2 + X3) / 3

            self.position[i, :] = np.clip(self.position[i, :], self.lower_bound, self.upper_bound)

    def update_scores(self):
        for i in range(self.n_agents):
            self.fitness[i] = objective_function(self.position[i, :])

            if self.fitness[i] < self.alpha_score:
                self.alpha_score = self.fitness[i]
                self.alpha_pos = self.position[i, :]

            elif self.fitness[i] < self.beta_score:
                self.beta_score = self.fitness[i]
                self.beta_pos = self.position[i, :]

            elif self.fitness[i] < self.delta_score:
                self.delta_score = self.fitness[i]
                self.delta_pos = self.position[i, :]

    def optimize(self):
        a_init = 2
        a_final = 0
        for t in range(self.max_iter):
            a = a_init - t * ((a_init - a_final) / self.max_iter)
            self.update_positions(a)
            self.update_scores()
            print(f"Iteration {t + 1}/{self.max_iter}, Alpha Score: {self.alpha_score}")
```

```
        return self.alpha_pos, self.alpha_score, self.beta_pos, self.beta_score, self.delta_pos,
self.delta_score

n_agents = 15          # Number of wolves (agents)
n_variables = 6       # Number of variables (dimensions)
max_iter = 50         # Maximum number of iterations
lower_bound = -10      # Lower bound of the search space
upper_bound = 10       # Upper bound of the search space

gwo = GreyWolfOptimization(n_agents, n_variables, max_iter, lower_bound, upper_bound)
best_position_alpha, best_score_alpha, best_position_beta, best_score_beta, best_position_delta,
best_score_delta = gwo.optimize()

print("\nBest Position (Alpha):", best_position_alpha)
print("Best Score (Alpha):", best_score_alpha)
print("\nBest Position (Beta):", best_position_beta)
print("Best Score (Beta):", best_score_beta)
print("\nBest Position (Delta):", best_position_delta)
print("Best Score (Delta):", best_score_delta)
```

OUTPUT-

```
Iteration 2/50, Alpha Score: 7.712442244400227
Iteration 3/50, Alpha Score: 7.712442244400227
Iteration 4/50, Alpha Score: 7.712442244400227
Iteration 5/50, Alpha Score: 1.9497872976004211
Iteration 6/50, Alpha Score: 0.7223276036534854
Iteration 7/50, Alpha Score: 0.5719621617497965
Iteration 8/50, Alpha Score: 0.5719621617497965
Iteration 9/50, Alpha Score: 0.5719621617497965
Iteration 10/50, Alpha Score: 0.3924613241989059
Iteration 11/50, Alpha Score: 0.10597247650884327
Iteration 12/50, Alpha Score: 0.10597247650884327
Iteration 13/50, Alpha Score: 0.10285665827120183
Iteration 14/50, Alpha Score: 0.08674503373485802
Iteration 15/50, Alpha Score: 0.08069748654418739
Iteration 16/50, Alpha Score: 0.06895588483253233
Iteration 17/50, Alpha Score: 0.049424233476573265
Iteration 18/50, Alpha Score: 0.03336928318259161
Iteration 19/50, Alpha Score: 0.03208916151105481
Iteration 20/50, Alpha Score: 0.028464329643102 7
Iteration 21/50, Alpha Score: 0.02216014293967526
Iteration 22/50, Alpha Score: 0.013469722132153528
Iteration 23/50, Alpha Score: 0.009719386279338252
Iteration 24/50, Alpha Score: 0.00450086374469863
Iteration 25/50, Alpha Score: 0.0037331905291921466
Iteration 26/50, Alpha Score: 0.0030185205204670577
Iteration 27/50, Alpha Score: 0.002169899479673297
Iteration 28/50, Alpha Score: 0.0015835866247275814
Iteration 29/50, Alpha Score: 0.0011595507122480872
Iteration 30/50, Alpha Score: 0.0011230385209238432
Iteration 31/50, Alpha Score: 0.0008003427919473188
Iteration 32/50, Alpha Score: 0.0007678140539387462
Iteration 33/50, Alpha Score: 0.0006326122222813632
Iteration 34/50, Alpha Score: 0.00047282016871396423
Iteration 35/50, Alpha Score: 0.00045178820815356443
Iteration 36/50, Alpha Score: 0.00042540714511105147
Iteration 37/50, Alpha Score: 0.0003499834996537306
Iteration 38/50, Alpha Score: 0.0002841722946158936
Iteration 39/50, Alpha Score: 0.00028074373148500534
Iteration 40/50, Alpha Score: 0.0002493845129698773
Iteration 41/50, Alpha Score: 0.0002218416451536955
Iteration 42/50, Alpha Score: 0.0002058390098409454
Iteration 43/50, Alpha Score: 0.00018476212189959764
Iteration 44/50, Alpha Score: 0.00018476212189959764
Iteration 45/50, Alpha Score: 0.00018024310473129253
Iteration 46/50, Alpha Score: 0.00016567415748705688
Iteration 47/50, Alpha Score: 0.00015917163775072481
Iteration 48/50, Alpha Score: 0.0001584549399180969
Iteration 49/50, Alpha Score: 0.00015731699026336652
Iteration 50/50, Alpha Score: 0.00015659736167240998

Best Position (Alpha): [ 0.00514578  0.00509858  0.00330332  0.0030153   0.00577133 -0.00712815]
Best Score (Alpha): 0.00015659736167240998

Best Position (Beta): [ 0.00510804  0.00519059  0.00331943  0.00302666  0.00577216 -0.00707785]
Best Score (Beta): 0.0001566274601897981

Best Position (Delta): [ 0.00511738  0.00516313  0.00333712  0.00302911  0.00581713 -0.00707403]
Best Score (Delta): 0.00015703831721392795
```