1. Initialization Phase

* Define the optimization problem as minimizing or maximizing an objective function , where d is a candidate solution in the *d*-dimensional search space.
* **Population Initialization**: Initialize a population of *N* rhinoceroses (candidate solutions), where each individual is a vector in the search space.

Here, *r* is a random vector uniformly distributed between 0 and 1, and ​ and ​ represent the lower and upper bounds of the solution space.

* Initialize parameters: Set the maximum number of iterations exploration-exploitation balance factor, and other algorithmic constants.

**Table.1. Algorithm Rhino Optimisation Algorithm Initiation**

|  |
| --- |
| **Algorithm:** ROA\_Initiation |
| Algorithm ROA\_Initiation:  Input: N, X\_min, X\_max, f(x), T\_max  Output: X\_initial, f(X\_initial), X\_best  Step 1: Define f(x) // Define the objective function  Step 2: Initialize N and D  Step 3: Initialize population:  for i = 1 to N do  for d = 1 to D do  X\_i^d = U(X\_min^d, X\_max^d) // Random initialization  end for  end for  Step 4: Evaluate fitness of each rhino:  for i = 1 to N do  f(X\_i) = evaluate(X\_i) // Compute fitness  end for  Step 5: Set X\_best = argmin(f(X)) // Best solution so far  Step 6: Set iteration t = 0  Return X\_initial, f(X\_initial), X\_best |

2. Fitness Evaluation and Solution Update

* **Fitness Function Evaluation**: Evaluate the fitness of each solution using the objective function
* **Selection**: Select the best-performing rhinoceros, i.e., the one with the lowest (or highest for maximization problems) objective function value:

=min {, , ..., }

* **Convergence Criteria**: The algorithm terminates if the maximum number of iterations is reached, or if the improvement in fitness is less than a threshold ϵ.

**Table 2**. Algorithm Rhino\_Fitness\_Evaluation\_and\_Solution\_Update

|  |
| --- |
| **Algorithm:** Fitness\_Evaluation\_and\_Solution\_Update Phase |
| Algorithm Rhino\_Fitness\_Evaluation\_and\_Solution\_Update(P, f, M, α, β):  Input:  P = {x\_1, x\_2, ..., x\_n} # Current population of solutions  f(x) # Objective fitness function  M # Momentum factor  α # Step size  β # Exploration factor  Output:  P' = {x\_1', x\_2', ..., x\_n'} # Updated population  x\_best' # Updated best solution  # Step 1: Initial Fitness Evaluation  x\_best = find\_best\_solution(P, f) # Find initial best solution based on fitness    # Step 2: Solution Update Rule  For each x\_i in P:  random\_factor = random\_uniform() # Generate random exploration factor  x\_i' = x\_i + M \* (x\_best - x\_i) + α \* β \* random\_factor # Update solution  # Step 3: Re-evaluate Fitness of Updated Solutions  For each x\_i' in P':  fitness\_new = f(x\_i') # Evaluate the fitness of updated solution  if fitness\_new is better than f(x\_best):  x\_best = x\_i' # Update the best solution  # Step 4: Return Updated Population and Best Solution  return P', x\_best |

3. Exploration Phase  
**Random Movement**: During the exploration phase, each rhinoceros moves in a randomized manner across the search space to explore potential regions.

= + .*.* *(* *- rand*

* is the position of rhinoceros *i* at iteration *t*.
* ​ is the best solution found so far.
* is the scaling factor for the influence of the best solution.
* is a random number in the range [0, 1] that controls the intensity of the exploration.
* *rand* is a random perturbation vector to enhance randomness in movement.
* controls the effect of the random perturbation on the solution's position.

**Table 4. Algorithm: Exploration Phase of Rhino Optimizer**

|  |
| --- |
| **Algorithm:** Exploration Phase of Rhino Optimizer |
| Begin:  1. For t = 1 to T do:  2. For each rhino i in the population (i = 1 to n):  3. Generate a random exploration factor E:  E = random\_uniform(0, 1)  4. If E < β:  5. Perform a wide exploration (global search):  ΔS\_explore = r\_max \* random\_uniform(-1, 1, dim)  S[i] = S[i] + ΔS\_explore  6. Evaluate the fitness of the new solution S[i]:  fitness\_current = Evaluate\_fitness(S[i])  7. If the new solution S[i] is better than the previous solution:  Update S[i] with the new position  8. If a convergence criterion is met (optional):  Exit loop early  9. End For (iterations)  10. Return updated population S'  End Algorithm |

4. Exploitation Phase

* Focused Foraging (Local Search): In this phase, rhinoceroses move towards promising regions where high-quality solutions have been found.

= . ( *( - + . M .* *(*

* α is the momentum factor that determines how aggressively the rhinoceroses converge towards
* *M* is the inertia coefficient controlling the rate of convergence.
* and are randomly selected individuals from the population (to introduce diversity).
* ζ is the weight assigned to this diversification term.

**Table 3. Algorithm: Exploitation Phase of Rhino Optimizer**

|  |
| --- |
| **Algorithm:** Exploitation Phase of Rhino Optimizer |
| Begin:  1. For t = 1 to T do:  2. For each rhino i in the population (i = 1 to n):  3. Compute the local search direction:  Δs = α \* (s\_best - S[i]) + d \* random\_uniform(-1, 1, dim)  4. Update the rhino's position:  S[i] = S[i] + Δs  5. Evaluate the fitness of the updated solution S[i]:  fitness\_current = Evaluate\_fitness(S[i])  fitness\_best = Evaluate\_fitness(s\_best)  6. If fitness\_current < fitness\_best (for minimization problems):  s\_best = S[i] # Update the best solution  7. If the convergence criterion is met (e.g., small change in fitness):  Exit loop early  8. End For (iterations)  9. Return updated population S' and best solution s\_best  End Algorithm |

5. Testing the Rhino Optimizer with SVM and Neural Networks

In this phase, the aim is to evaluate the performance of the Rhino Optimizer (ROA) by integrating it with Support Vector Machines (SVM) and Neural Networks (NN). The steps for testing are as follows.

1. Data Preprocessing:

* Input: Dataset (Training and Testing sets)
* Action:
  + Normalize or standardize the dataset.
  + Split the dataset into training and testing sets.
  + Apply necessary preprocessing (handling missing values, encoding categorical variables).
* Output: Preprocessed dataset

2. Define SVM and Neural Network Models:

* Define the structure of the Support Vector Machine (SVM) and the Neural Network.
* For SVM, parameters to be optimized include the kernel type, regularization parameter (C), and gamma.
* For Neural Networks, parameters include the number of layers, number of neurons in each layer, activation function, learning rate, and dropout rate.

3. Initialize Rhino Optimizer:

* Input: Predefined population size, number of iterations, and bounds for hyperparameters (for SVM and NN).
* Initialize the population of solutions (randomly generated sets of hyperparameters).
* Set up the Rhino Optimizer with predefined parameters for exploration and exploitation.

4. Optimization Process:

* Input: Initialized population, SVM, and NN models.
  1. For each hyperparameter set (solution) in the population:
     + Apply the current hyperparameters to the SVM and Neural Network models.
     + Train the SVM and NN models using the training dataset.
     + Evaluate the performance (e.g., accuracy, F1-score) on the testing dataset.
  2. Use the fitness (model performance) to update the solutions during the exploration and exploitation phases.
  3. After a number of iterations, select the best-performing hyperparameter set.

5. Training with Optimized Hyperparameters:

* Input: Best hyperparameter sets for SVM and NN (from the Rhino Optimizer).
* Retrain the SVM and Neural Network models using the optimized hyperparameters on the entire training dataset.
* Evaluate the models on the testing dataset.
* Output: Performance metrics (accuracy, precision, recall, F1-score) for both SVM and NN with optimized hyperparameters.

6. Performance Comparison:

* Compare the performance of the models before and after hyperparameter optimization.
* Compare the performance of SVM and Neural Networks when optimized using the Rhino Optimizer.

7. Analysis and Conclusion:

* Analyse the improvements in model performance due to hyperparameter tuning with the Rhino Optimizer.
* Draw conclusions regarding the effectiveness of ROA in optimizing hyperparameters for SVM and Neural Networks.