Practical No: 05

Title: Design and implement Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.
Problem Statement: Design and implement the spray drying process of coconut milk involves complex interactions of multiple parameters that affect the quality of the dried product.
Prerequisite:
Basics of Python
Software Requirements: Jupyter
Hardware Requirements:
Intel Core I3, 4GB RAM, 500 GB HDD

Learning Objectives:

1. To optimize the parameters of the genetic algorithm in a hybrid genetic algorithm-neural network model for the spray drying process of coconut milk.
2. To improve the accuracy and efficiency of the modeling process by tuning the genetic algorithm parameters.
3. To demonstrate the effectiveness of the optimized hybrid model in predicting the quality of the dried coconut milk.

Outcomes:

1. Identification of the optimal parameters for the genetic algorithm in the hybrid model.
2. Improved prediction accuracy and efficiency of the hybrid model compared to traditional modeling approaches.

Theory:

Modeling complex processes like spray drying coconut milk presents a multi-faceted challenge. Nonlinear relationships and numerous influential factors demand sophisticated methods beyond traditional linear models. While Artificial Neural Networks (ANNs) excel in capturing such nonlinearities, they can be prone to initialization issues and require substantial data for optimal performance. Genetic Algorithms (GAs), on the other hand, adeptly navigate global optimization landscapes but can be computationally expensive. This research explores the intriguing avenue of optimizing GA parameters in a hybrid GA-NN approach to model spray drying, aiming to synergistically leverage the strengths of both techniques while mitigating their limitations.

Our endeavor is driven by three key objectives:

1. Fine-tuning GA Parameters: We seek to identify a set of GA parameters that efficiently navigate the solution space, ultimately leading to optimized network weights for modeling spray drying. Population size, crossover rate, and mutation rate are prominent parameters we intend to carefully calibrate.

2. Elevating Model Accuracy: Our ambition is to develop a model that surpasses the accuracy of both standalone ANNs and GA-NN models using unoptimized parameters. This enhanced

model should faithfully represent the intricate dynamics of the spray drying process, offering superior predictive capabilities.

3. Striking a Balance: Achieving high accuracy is commendable, but it shouldn't come at the expense of exorbitant computational costs. We delve into the interplay between accuracy and computation time, striving to find GA parameter settings that strike a favorable balance between thorough exploration and focused exploitation, minimizing execution time without compromising model efficacy.

Through this endeavor, we aspire to achieve two significant outcomes:

1. A Champion Model: The culmination of our efforts will be a meticulously optimized GA-NN model capable of precisely predicting spray drying performance based on diverse input parameters. This model should stand out from its peers, demonstrating superior accuracy and robustness.

2. Performance Benchmarking: We will meticulously quantify the model's improved accuracy and the reduction in computational cost compared to alternative approaches. This comparative analysis will illuminate the effectiveness of our optimization strategy, providing valuable insights and paving the way for further advancements.

The optimization process typically involves the following steps:
Parameter Initialization: Initialize the genetic algorithm parameters, including population size, crossover rate, mutation rate, and selection strategy.
Fitness Evaluation: Evaluate the fitness of each individual in the population using the NN model. The fitness function is usually based on the error between predicted and actual drying parameters.
Selection: Select individuals from the population based on their fitness scores to form the next generation.
Crossover and Mutation: Apply crossover and mutation operators to create offspring for the next generation.
Replacement: Replace the current population with the new generation of individuals.
Termination: Repeat the process until a termination condition is met, such as reaching a maximum number of generations or achieving a desired level of convergence.

By optimizing the genetic algorithm parameters in the hybrid GA-NN model, we expect to improve the accuracy of predicting the quality of dried coconut milk powder. This optimized model can provide valuable insights for optimizing the spray drying process and enhancing the quality of the final product.
Here's an example of how you can optimize the genetic algorithm parameters in a hybrid GA-NN model for predicting the quality of dried coconut milk powder using Python. This example uses **the genetic_algorithm** library for the genetic algorithm part and **scikit-learn** for the neural network part. Please note that this is a simplified example for demonstration purposes and may require further customization for your specific application.

Conclusion: Thus Designed and Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.

```
Optimization of genetic algorithm parameter in hybrid genetic algorithm-
neural network modelling:
Application to spray drying of coconut milk.
```

In [1]:
```
pip install deap
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting deap
  Downloading deap-1.4.1-cp39-cp39-win_amd64.whl (109 kB)
     ------------------------------------ 109.9/109.9 kB 491.0 kB/s eta 0:00:
00
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-pac
kages (from deap) (1.21.5)
Installing collected packages: deap
Successfully installed deap-1.4.1
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import random
from deap import base, creator, tools, algorithms
```

In [3]:
```
# Define evaluation function (this is a mock function, replace this with your
def evaluate(individual):
    # Here 'individual' represents the parameters for the neural network
    # You'll need to replace this with your actual evaluation function that tr
    # Return a fitness value (here, a random number is used as an example)
    return random.random(),
```

In [4]:
```
# Define genetic algorithm parameters
POPULATION_SIZE = 10
GENERATIONS = 5
```

In [5]:
```
# Create types for fitness and individuals in the genetic algorithm
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
```

In [6]:
```
# Initialize toolbox
toolbox = base.Toolbox()
```

In [7]:
```
# Define attributes and individuals
toolbox.register("attr_neurons", random.randint, 1, 100)  # Example: number of
toolbox.register("attr_layers", random.randint, 1, 5)  # Example: number of la
toolbox.register("individual", tools.initCycle, creator.Individual, (toolbox.a
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

In [8]:
```
# Genetic operators
toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=1, up=100, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
```

In [9]:
```
# Create initial population
population = toolbox.population(n=POPULATION_SIZE)
```

In [10]:
```
# Run the genetic algorithm
for gen in range(GENERATIONS):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)

    fitnesses = toolbox.map(toolbox.evaluate, offspring)
    for ind, fit in zip(offspring, fitnesses):
        ind.fitness.values = fit

    population = toolbox.select(offspring, k=len(population))
```

In [11]:
```
# Get the best individual from the final population
best_individual = tools.selBest(population, k=1)[0]
best_params = best_individual
```

In [12]:
```
# Print the best parameters found
print("Best Parameters:", best_params)
```

```
Best Parameters: [51, 4]
```

In [ ]: