+ Code + Text

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

```
import random
!pip install deap
from deap import base, creator, tools, algorithms
```

```
Collecting deap
  Downloading deap-1.4.2-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from deap) (2.0.2)
Downloading deap-1.4.2-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (135 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 2.5 MB/s eta 0:00:00
Installing collected packages: deap
Successfully installed deap-1.4.2
```

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


# Define genetic algorithm parameters
POPULATION_SIZE = 10
GENERATIONS = 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)


# Initialize toolbox
toolbox = base.Toolbox()


# Define attributes and individuals
toolbox.register("attr_neurons", random.randint, 1, 100)  # Example: number of neurons
toolbox.register("attr_layers", random.randint, 1, 5)  # Example: number of layers
toolbox.register("individual", tools.initCycle, creator.Individual, (toolbox.attr_neurons, toolbox.attr_layers), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)


# 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)


# Create initial population
population = toolbox.population(n=POPULATION_SIZE)


# 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))


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


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

```
Best Parameters: [25, 2]
```

Start coding or generate with AI.