Assignment No. 4

Title: Optimization of Machine Learning Model Parameters using Genetic Algorithm

Objectives:

1. To understand the concept of Genetic Algorithms and their role in optimization.
2. To apply GA for tuning hyperparameters of a machine learning model.
3. To compare GA-based optimization with traditional approaches like grid search or random search.
4. To evaluate the performance improvement of the ML model after optimization.

Theory:

A Genetic Algorithm (GA) is a nature-inspired optimization technique based on the principles of evolutionary biology and natural selection. The idea was first introduced by John Holland in the 1970s and has since been widely adopted for solving both simple and complex optimization problems. Unlike conventional optimization techniques, GA does not require gradient information or mathematical properties of the problem, making it suitable for non-linear, high-dimensional, and discontinuous search spaces.

At its core, GA imitates the process of Darwinian evolution, where stronger individuals in a population are more likely to survive and pass on their traits to the next generation. This is applied in computational systems to iteratively evolve solutions to optimization problems.

1. Representation (Chromosomes)

Each potential solution is encoded as a chromosome, often represented as a binary string, integer, or real-valued vector. For example, in machine learning hyperparameter tuning, a chromosome might encode values such as:

- Learning rate = 0.01
- Number of hidden layers = 3
- Regularization parameter = 0.1

2. Population Initialization

The algorithm starts with a population of randomly generated chromosomes. The population size is an important parameter, as it determines the diversity of potential solutions in the search space. A larger population increases exploration but requires more computation.

3. Fitness Evaluation

Each chromosome is evaluated using a fitness function, which measures the quality of the solution. In the case of machine learning, this may correspond to the accuracy, precision, recall, or mean squared error of the model trained with the given parameters. The better the model performs, the higher the fitness value of its corresponding chromosome.

4. Selection

The next step is to select parents from the population based on their fitness. Higher fitness solutions have a greater probability of being selected. Common strategies include:

- Roulette Wheel Selection (Proportionate Selection): Probability proportional to fitness.

- Tournament Selection: A subset of chromosomes competes, and the best is selected.
- Rank Selection: Chromosomes are ranked, and probabilities are assigned accordingly.

This ensures that good solutions are more likely to reproduce while maintaining diversity.

5. Crossover (Recombination)

Selected parents undergo crossover to produce offspring by exchanging parts of their chromosomes. This mimics biological reproduction and helps combine the strengths of two solutions. Types of crossover include:

- Single-Point Crossover: A crossover point is chosen, and segments are swapped.
- Two-Point Crossover: Two crossover points are chosen.
- Uniform Crossover: Each gene is swapped with some probability.

This mechanism introduces new solutions that can potentially perform better than their parents.

6. Mutation

To maintain diversity in the population and prevent premature convergence, GA introduces small random changes to chromosomes. For example:

- Flipping a bit in a binary string.
- Slightly altering a real-valued parameter.

Mutation ensures that the search process explores new regions of the solution space.

7. Replacement (Survivor Selection)

After generating offspring, the new generation is formed by replacing some or all members of the old population. Strategies include:

- Generational Replacement: Entire population replaced.
- Elitism: Best individuals from the previous generation are carried forward.

8. Termination Criteria

The GA continues until a stopping condition is met, such as:

- A maximum number of generations.
- No significant improvement in fitness.
- Achieving a desired accuracy or error threshold.

Application in Machine Learning

In machine learning, selecting optimal hyperparameters is often challenging due to the curse of dimensionality and non-linear interactions between parameters. Traditional methods like grid search or random search are computationally expensive and often miss good solutions.

By contrast, GA offers the following advantages:

- **Exploration:** Searches a wide space of possible parameter values.
- **Adaptation:** Learns from previous generations to improve solutions.
- **Efficiency:** Finds near-optimal solutions with fewer evaluations than exhaustive search.

For instance, in optimizing a Support Vector Machine (SVM):

- The chromosome may encode kernel type, C, and gamma.
- GA evaluates each combination using cross-validation accuracy.
- Over generations, the algorithm converges toward the best set of parameters.

Conclusion:

This assignment demonstrates how Genetic Algorithms can optimize machine learning model parameters efficiently. GA provides a robust, adaptive, and intelligent search strategy for hyperparameter tuning, outperforming conventional methods like grid search in terms of flexibility and scalability.

By applying GA, the machine learning model achieves higher accuracy and generalization, proving the effectiveness of evolutionary computing in modern AI applications.

Code:

```
import numpy as np
import random
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# --- Genetic Algorithm Setup ---
POP_SIZE = 20      # number of individuals
N_GENERATIONS = 10 # iterations
MUTATION_RATE = 0.2

# Chromosome: [max_depth, min_samples_split]
def create_chromosome():
    return [random.randint(1, 20), random.randint(2, 10)]

def fitness(chromosome):
    max_depth, min_samples_split = chromosome
    model = DecisionTreeClassifier(max_depth=max_depth,
                                   min_samples_split=min_samples_split)
    scores = cross_val_score(model, X, y, cv=5)
    return scores.mean()

def selection(population, fitnesses):
    idx = np.argsort(fitnesses)[-2:]  # select best two
    return [population[idx[0]], population[idx[1]]]

def crossover(parent1, parent2):
    point = random.randint(0, len(parent1)-1)
    child1 = parent1[:point] + parent2[point:]
    child2 = parent2[:point] + parent1[point:]
    return child1, child2

def mutate(chromosome):
    if random.random() < MUTATION_RATE:
        chromosome[0] = random.randint(1, 20)
    if random.random() < MUTATION_RATE:
        chromosome[1] = random.randint(2, 10)
    return chromosome
```

```
# --- Run GA ---
population = [create_chromosome() for _ in range(POP_SIZE)]

for gen in range(N_GENERATIONS):
    fitnesses = [fitness(chromo) for chromo in population]
    print(f"Generation {gen} - Best Fitness: {max(fitnesses):.4f}")

    new_population = []
    parents = selection(population, fitnesses)
    for _ in range(POP_SIZE // 2):
        child1, child2 = crossover(parents[0], parents[1])
        new_population.append(mutate(child1))
        new_population.append(mutate(child2))

    population = new_population

# Best result
fitnesses = [fitness(chromo) for chromo in population]
best_idx = np.argmax(fitnesses)
print("Best Hyperparameters:", population[best_idx])
print("Best Accuracy:", fitnesses[best_idx])
```

Output:

Generation 0 - Best Fitness: 0.9733
Generation 1 - Best Fitness: 0.9733
Generation 2 - Best Fitness: 0.9733
Generation 3 - Best Fitness: 0.9733
Generation 4 - Best Fitness: 0.9733
Generation 5 - Best Fitness: 0.9733
Generation 6 - Best Fitness: 0.9733
Generation 7 - Best Fitness: 0.9733
Generation 8 - Best Fitness: 0.9733
Generation 9 - Best Fitness: 0.9733
Best Hyperparameters: [3, 4]
Best Accuracy: 0.9733333333333334