LAB 1

Genetic Algorithm for Optimization Problems:

Application :

Optimization of a neural network for binary classification

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Helper functions for the Neural Network
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def forward_pass(X, weights1, weights2):
    hidden_input = np.dot(X, weights1)
    hidden_output = sigmoid(hidden_input)
    output_input = np.dot(hidden_output, weights2)
    output = sigmoid(output_input)
    return output

def compute_fitness(weights, X_train, y_train):
    predictions = forward_pass(X_train, weights['w1'], weights['w2'])
    predictions = (predictions > 0.5).astype(int)
    accuracy = accuracy_score(y_train, predictions)
    return accuracy

# User input for dataset creation
n_samples = int(input("Enter the number of samples: "))
n_features = int(input("Enter the number of features: "))
test_size = float(input("Enter the test size (e.g., 0.2 for 20%): "))

# Generate dataset
X, y = make_classification(n_samples=n_samples, n_features=n_features,
n_informative=int(n_features * 0.8), n_classes=2)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=test_size)

# Neural Network Parameters
input_size = X.shape[1]
hidden_size = int(input("Enter the number of neurons in the hidden layer: "))
output_size = 1

# GA Parameters
```

```
population_size = int(input("Enter the population size: "))
generations = int(input("Enter the number of generations: "))
mutation_rate = float(input("Enter the mutation rate (e.g., 0.1 for 10%): "))

# Initialize Population
population = []
for _ in range(population_size):
    individual = {
        'w1': np.random.randn(input_size, hidden_size),
        'w2': np.random.randn(hidden_size, output_size)
    }
    population.append(individual)

# Tracking performance
best_fitness_history = []
average_fitness_history = []

# Main Genetic Algorithm Loop
for generation in range(generations):
    # Evaluate Fitness of each Individual
    fitness_scores = np.array([compute_fitness(individual, X_train, y_train) for
individual in population])
    best_fitness = np.max(fitness_scores)
    average_fitness = np.mean(fitness_scores)
    best_fitness_history.append(best_fitness)
    average_fitness_history.append(average_fitness)

    # Selection: Select top half of the population
    sorted_indices = np.argsort(fitness_scores)[::-1]
    population = [population[i] for i in sorted_indices[:population_size // 2]]

    # Crossover and Mutation
    new_population = []
    while len(new_population) < population_size:
        parents = np.random.choice(population, 2, replace=False)
        child = {
            'w1': (parents[0]['w1'] + parents[1]['w1']) / 2,
            'w2': (parents[0]['w2'] + parents[1]['w2']) / 2
        }

        # Mutation
        if np.random.rand() < mutation_rate:
            child['w1'] += np.random.randn(*child['w1'].shape) * 0.1
            child['w2'] += np.random.randn(*child['w2'].shape) * 0.1

        new_population.append(child)

    population = new_population
```

```
    print(f"Generation {generation + 1}, Best Fitness: {best_fitness:.4f}")

# Evaluate the best individual on validation set
best_individual = population[np.argmax(fitness_scores)]
predictions = forward_pass(X_val, best_individual['w1'], best_individual['w2'])
predictions = (predictions > 0.5).astype(int)
final_accuracy = accuracy_score(y_val, predictions)
print(f"Final Accuracy on Validation Set: {final_accuracy:.4f}")

# Plotting the results
plt.figure(figsize=(10, 5))
plt.plot(best_fitness_history, label='Best Fitness')
plt.plot(average_fitness_history, label='Average Fitness')
plt.title('Fitness Over Generations')
plt.xlabel('Generation')
plt.ylabel('Fitness')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Enter the number of samples: 500
Enter the number of features: 10
Enter the test size (e.g., 0.2 for 20%): 0.2
Enter the number of neurons in the hidden layer: 5
Enter the population size: 20
Enter the number of generations: 50
Enter the mutation rate (e.g., 0.1 for 10%): 0.1
Generation 1, Best Fitness: 0.6725
Generation 2, Best Fitness: 0.7925
Generation 3, Best Fitness: 0.7325
Generation 4, Best Fitness: 0.7750
Generation 5, Best Fitness: 0.7750
Generation 6, Best Fitness: 0.7650
Generation 7, Best Fitness: 0.7575
Generation 8, Best Fitness: 0.7550
Generation 9, Best Fitness: 0.7800
Generation 10, Best Fitness: 0.7675
Generation 11, Best Fitness: 0.7650
Generation 12, Best Fitness: 0.7675
Generation 13, Best Fitness: 0.7800
Generation 14, Best Fitness: 0.7825
Generation 15, Best Fitness: 0.7725
Generation 16, Best Fitness: 0.7725
Generation 17, Best Fitness: 0.7850
Generation 18, Best Fitness: 0.7800
Generation 19, Best Fitness: 0.7800
Generation 20, Best Fitness: 0.7800
Generation 21, Best Fitness: 0.7850
Generation 22, Best Fitness: 0.7850
```

```
Generation 23, Best Fitness: 0.7850
Generation 24, Best Fitness: 0.7825
Generation 25, Best Fitness: 0.7825
Generation 26, Best Fitness: 0.7800
Generation 27, Best Fitness: 0.8000
Generation 28, Best Fitness: 0.7900
Generation 29, Best Fitness: 0.7850
Generation 30, Best Fitness: 0.7925
Generation 31, Best Fitness: 0.7925
Generation 32, Best Fitness: 0.7925
Generation 33, Best Fitness: 0.7925
Generation 34, Best Fitness: 0.7900
Generation 35, Best Fitness: 0.7900
Generation 36, Best Fitness: 0.7925
Generation 37, Best Fitness: 0.7900
Generation 38, Best Fitness: 0.7900
Generation 39, Best Fitness: 0.7900
Generation 40, Best Fitness: 0.7900
Generation 41, Best Fitness: 0.7950
Generation 42, Best Fitness: 0.7925
Generation 43, Best Fitness: 0.8100
Generation 44, Best Fitness: 0.8050
Generation 45, Best Fitness: 0.8050
Generation 46, Best Fitness: 0.8075
Generation 47, Best Fitness: 0.8075
Generation 48, Best Fitness: 0.8075
Generation 49, Best Fitness: 0.8075
Generation 50, Best Fitness: 0.8125
Final Accuracy on Validation Set: 0.7600
```