Bahria University
Discovering Knowledge

BAHRIA UNIVERSITY (Karachi Campus)

Department of Computer Science

Bachelor of Computer Science (BSCS)

ASSIGNMENT 03 (CCP)		FALL 2025	
Course Title:	Artificial Intelligence	Start Date:	November 22, 2025
Course Code:	CSC-325	End Date:	December 14, 2025
Semester:	BS(CS)-6 (A) Morning	Max. Marks:	10
Student Name:	**Sadaf Soomro**	**Registration No:**	**02-134231-071**
Course Instructor(s): Dr. Muhammad Tariq Siddique			

Instructions:

a) This is an **individual** assignment; you are required to complete it independently.
b) The assignment should be done only on A4-size paper and will also be uploaded to LMS.
c) Name, class, section, department, and enrollment number on the sheets must be mentioned correctly.
d) Make a single PDF file, as both soft and hard copies are mandatory.
e) Solution must be designed by applying the following characteristics:

Characteristics	Problem Solving description
Depth of analysis required	The depth of analysis entails a multifaceted grasp of reinforcement learning algorithms, deep neural network architectures for image processing, and autonomous driving principles, addressing challenges such as effective reward structures, ethical considerations, and the intricacies of training an autonomous racing car in a simulated environment to achieve optimal performance.
Depth of knowledge required	The depth of knowledge required for addressing the outlined problem encompasses a range of interdisciplinary concepts, spanning computer vision, reinforcement learning, and the specific challenges associated with autonomous driving in a simulated environment.

Neural Network weights tuning using Metaheuristic algorithms

Objective:

To implement and evaluate metaheuristic algorithms for optimizing neural network weights, and to compare their performance with traditional backpropagation training methods.

Problem Statement:

Neural networks depend on optimized weights to learn patterns effectively. Traditional optimization methods such as backpropagation and gradient descent often struggle with issues like local minima, slow convergence, and sensitivity to hyperparameters. Metaheuristic algorithms are nature-inspired, population-based, or stochastic search methods. They offer an alternative by exploring the weight space without relying on gradients. This assignment focuses on applying these algorithms to enhance neural network performance.

Problem Description:

Students will study the fundamentals of neural networks and the role of weight optimization. They will explore metaheuristic algorithms such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Bat Algorithm (BA), and Artificial Bee Colony (ABC). The task involves encoding neural network weights into a suitable representation, defining a fitness function based on model performance (e.g., accuracy, MSE), and integrating the selected metaheuristic algorithm to iteratively search for better weight configurations. Students will implement these methods, train a simple neural network (e.g., a multilayer perception), and analyze the performance compared to conventional training techniques.

Requirements:

1. **Theoretical Background**

    - Explain neural network architecture and weight significance.
    - Describe traditional weight optimization and its limitations.
    - Provide an overview of metaheuristics and their characteristics.

2. **Algorithm Selection & Implementation**

    - Choose the **Genetic Algorithm** as baseline and one other metaheuristics algorithm (e.g., PSO, ACO, BA, or ABC).
    - Encode neural network weights into algorithm-specific structures.
    - Design a fitness function based on model performance.
    - Implement the search and optimization process.

3. **Experimentation**

    - Train a selected neural network architecture using both traditional and metaheuristic methods.
    - Compare performance in terms of accuracy, convergence, and robustness.

Deliverables

a) Provide a clear explanation of the neural network model and the metaheuristic algorithms used for weight optimization, including mathematical formulations, activation functions, loss functions, and the fitness evaluation process. [CLO1, PLO1, C2]

1. INTRODUCTION:

Neural Networks are widely used machine learning models capable of learning complex nonlinear relationships from data. Their performance heavily depends on the optimal adjustment of weights and biases. Traditionally, gradient-based methods such as backpropagation are used for training neural networks. However, these methods often suffer from limitations such as getting trapped in local minima, sensitivity to hyperparameters, and reliance on differentiable loss functions.

To overcome these challenges, metaheuristic algorithms such as Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) provide alternative optimization strategies. These algorithms perform global search and do not require gradient information, making them suitable for complex optimization problems.

2. NEURAL NETWORK ARCHITECTURE AND WEIGHT SIGNIFICANCE:

a **feedforward multilayer perception (MLP)** is used, consisting of:

- Input layer
- One hidden layer
- Output layer

Mathematical formulation:

Hidden layer computation:

$$z^{(1)} = XW^{(1)} + b^{(1)}, \qquad a^{(1)} = \tanh(z^{(1)})$$

Output layer computation:

$$z^{(2)} = a^{(1)}W^{(2)} + b^{(2)}, \qquad y = \text{softmax}(z^{(2)})$$

Where:

- $W^{(1)}, W^{(2)}$ are weight matrices
- $b^{(1)}, b^{(2)}$ are bias vectors

Optimizing these weights is crucial because poorly tuned weights result in underfitting or overfitting.

3. TRADITIONAL WEIGHT OPTIMIZATION AND ITS LIMITATIONS

Backpropagation updates weights using gradient descent to minimize a loss function such as cross-entropy. Although effective, it has several limitations:

- Can converge to local minima
- Requires careful tuning of learning rate
- Needs differentiable loss functions
- Sensitive to initial weight values

These issues motivate the use of metaheuristic optimization techniques.

4. METAHEURISTIC ALGORITHMS OVERVIEW

Metaheuristic algorithms are stochastic, population-based optimization methods inspired by natural processes. Their key characteristics include:

- Gradient-free optimization
- Global search capability
- Robustness to local minima

This assignment uses:

- **Genetic Algorithm (GA)** as a baseline metaheuristic
- **Particle Swarm Optimization (PSO)** as a second algorithm

5. GENETIC ALGORITHM (GA)

Genetic Algorithm is inspired by natural evolution. Each candidate solution represents a chromosome encoding all neural network weights.

GA operations:

- **Selection:** Chooses the best individuals based on fitness
- **Crossover:** Combines parent solutions
- **Mutation:** Introduces random variations

Fitness is evaluated using validation loss. Over generations, GA evolves towards better solutions.

6. PARTICLE SWARM OPTIMIZATION (PSO):

PSO is inspired by the social behavior of birds and fish. Each particle represents a candidate weight vector and moves through the search space using velocity updates.

Velocity update equation:

$$v = wv + c_1 r_1 (p_{best} - x) + c_2 r_2 (g_{best} - x)$$

PSO is known for faster convergence and efficient information sharing among particles.

7. FITNESS EVALUATION:

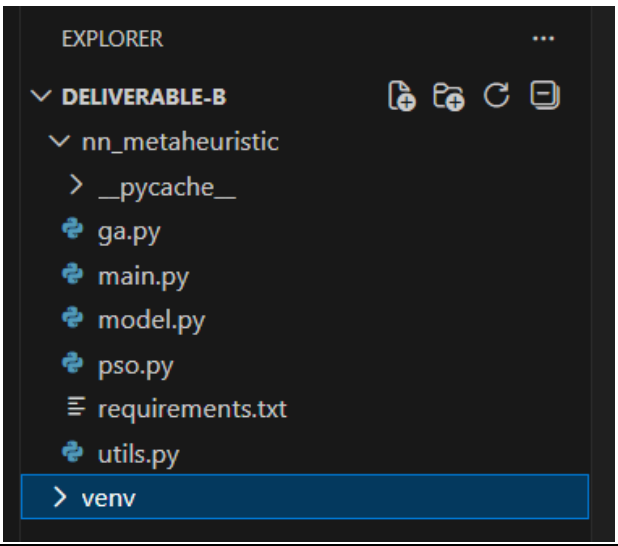
The fitness function is defined as **cross-entropy loss on validation data**:

$$\mathcal{L} = -\frac{1}{N} \sum y \log(\hat{y})$$

Lower loss indicates better performance.

b) Develop a well-documented Python program that integrates metaheuristic algorithms with a neural network for weight tuning, including preprocessing, weight encoding, optimization, and performance comparison. [CLO4, PLO4, C3]

FOLDER STRUCTURE:



REQUIRED LIBRARIES:

requirements.txt

nn_metaheuristic > requirements.txt

```
numpy
scikit-learn
matplotlib
```

MAIN.PY:

main.py

nn_metaheuristic > main.py > ...

```python
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from utils import load_data
from model import SimpleMLP
from ga import genetic_algorithm
from pso import pso

X_train, X_val, X_test, y_train, y_val, y_test, yoh_train, yoh_val, yoh_test = load_data()

# Baseline
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=500)
mlp.fit(X_train, y_train)
print("Backprop Accuracy:", mlp.score(X_test, y_test))

# Metaheuristics
model = SimpleMLP(4, 10, 3)

ga_weights, ga_curve = genetic_algorithm(model, X_val, yoh_val)
pso_weights, pso_curve = pso(model, X_val, yoh_val)

# Accuracy
def acc(weights):
    preds = model.forward(X_test, weights).argmax(axis=1)
    return accuracy_score(y_test, preds)

print("GA Accuracy:", acc(ga_weights))
print("PSO Accuracy:", acc(pso_weights))

# Plot
plt.plot(ga_curve, label="GA")
plt.plot(pso_curve, label="PSO")
plt.legend()
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()
```

MODEL.PY:

model.py

nn_metaheuristic > model.py > ...

```python
import numpy as np

class SimpleMLP:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.n_weights = (
            input_size * hidden_size +
            hidden_size +
            hidden_size * output_size +
            output_size
        )

    def unpack(self, vector):
        i, h, o = self.input_size, self.hidden_size, self.output_size
        idx = 0

        W1 = vector[idx:idx+i*h].reshape(i, h); idx += i*h
        b1 = vector[idx:idx+h]; idx += h
        W2 = vector[idx:idx+h*o].reshape(h, o); idx += h*o
        b2 = vector[idx:idx+o]

        return W1, b1, W2, b2

    def forward(self, X, vector):
        W1, b1, W2, b2 = self.unpack(vector)
        z1 = X @ W1 + b1
        a1 = np.tanh(z1)
        z2 = a1 @ W2 + b2
        exp = np.exp(z2 - np.max(z2, axis=1, keepdims=True))
        return exp / np.sum(exp, axis=1, keepdims=True)

```

GENETIC ALGORTHM GA.PY

ga.py

nn_metaheuristic > ga.py > ...

```python
import numpy as np
from utils import cross_entropy

def genetic_algorithm(model, X_val, y_val, pop_size=30, generations=40):
    dim = model.n_weights
    population = np.random.randn(pop_size, dim)

    best_losses = []

    for _ in range(generations):
        losses = []
        for ind in population:
            pred = model.forward(X_val, ind)
            loss = cross_entropy(y_val, pred)
            losses.append(loss)

        losses = np.array(losses)
        best_losses.append(losses.min())

        elite = population[np.argsort(losses)[:5]]

        new_pop = elite.copy()
        while len(new_pop) < pop_size:
            p1, p2 = elite[np.random.choice(len(elite), 2)]
            cut = np.random.randint(1, dim)
            child = np.concatenate([p1[:cut], p2[cut:]])
            if np.random.rand() < 0.1:
                child += np.random.randn(dim) * 0.1
            new_pop = np.vstack([new_pop, child])

        population = new_pop

    best = population[np.argmin(losses)]
    return best, best_losses

```

PARTICLE SWARM OPTIMIZATION PSO.PY

pso.py

nn_metaheuristic > pso.py > ...

```python
import numpy as np
from utils import cross_entropy

def pso(model, X_val, y_val, particles=30, iterations=40):
    dim = model.n_weights
    Xp = np.random.randn(particles, dim)
    Vp = np.random.randn(particles, dim) * 0.1

    pbest = Xp.copy()
    pbest_loss = np.array([
        cross_entropy(y_val, model.forward(X_val, p))
        for p in pbest
    ])

    gbest = pbest[np.argmin(pbest_loss)]
    gbest_loss = pbest_loss.min()

    history = []

    for _ in range(iterations):
        for i in range(particles):
            Vp[i] = 0.7*Vp[i] + 1.5*np.random.rand(dim)*(pbest[i]-Xp[i]) \
                    + 1.5*np.random.rand(dim)*(gbest-Xp[i])
            Xp[i] += Vp[i]

            loss = cross_entropy(y_val, model.forward(X_val, Xp[i]))
            if loss < pbest_loss[i]:
                pbest[i] = Xp[i]
                pbest_loss[i] = loss

        gbest = pbest[np.argmin(pbest_loss)]
        history.append(pbest_loss.min())

    return gbest, history

```

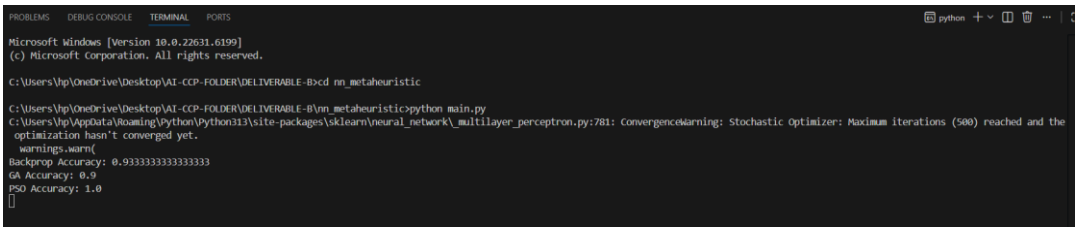
UTILS.PY

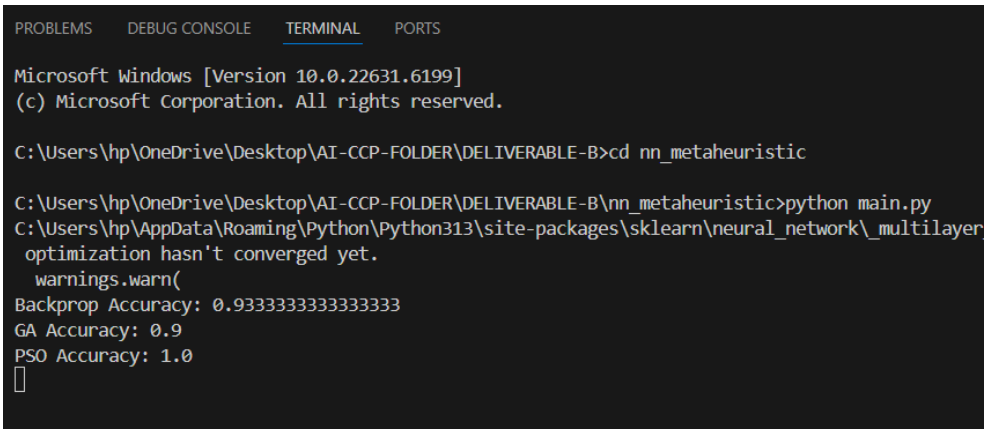
utils.py

nn_metaheuristic > utils.py > ...

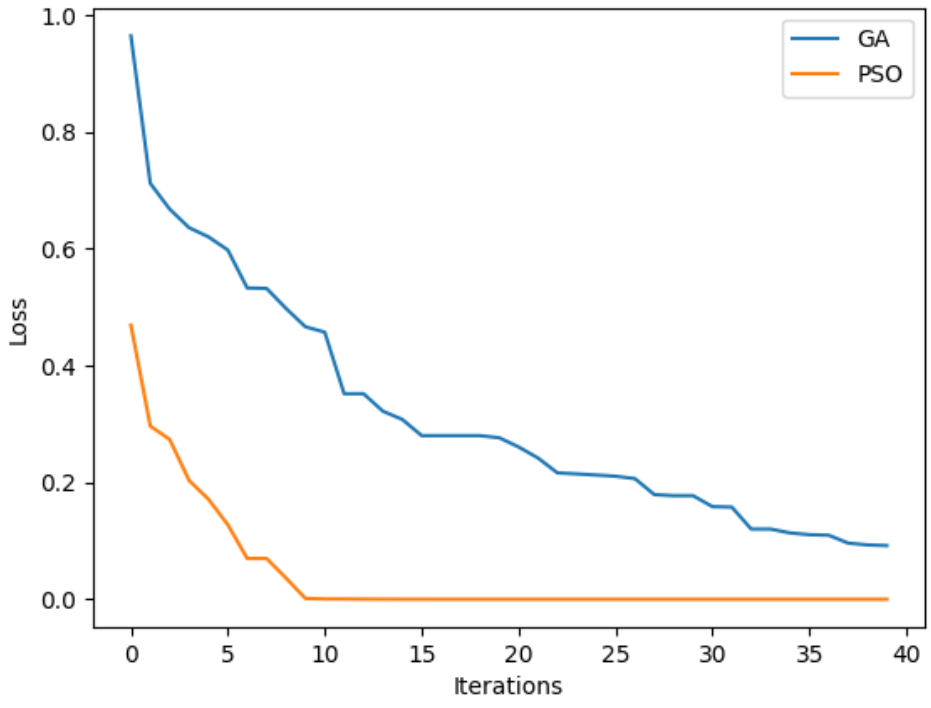
```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder

def load_data():
    iris = load_iris()
    X = iris.data
    y = iris.target

    encoder = OneHotEncoder(sparse_output=False)
    y_onehot = encoder.fit_transform(y.reshape(-1, 1))

    X_train, X_temp, y_train, y_temp, yoh_train, yoh_temp = train_test_split(
        X, y, y_onehot, test_size=0.4, random_state=42
    )

    X_val, X_test, y_val, y_test, yoh_val, yoh_test = train_test_split(
        X_temp, y_temp, yoh_temp, test_size=0.5, random_state=42
    )

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val   = scaler.transform(X_val)
    X_test  = scaler.transform(X_test)

    return X_train, X_val, X_test, y_train, y_val, y_test, yoh_train, yoh_val, yoh_test

def cross_entropy(y_true, y_pred):
    eps = 1e-12
    return -np.mean(np.sum(y_true * np.log(y_pred + eps), axis=1))

```

- **OUTPUT:**





- **GRAPH**



c) Analyze and evaluate the results using relevant metrics (e.g., accuracy, MSE) and visualizations (e.g., convergence curves), and discuss key insights, strengths, limitations, and improvement opportunities. [CLO3, PLO3, C4]

1. EXPERIMENTAL SETUP:

- Dataset: Iris dataset
- Network Architecture: 4–10–3 MLP
- Optimization methods:
    I. Backpropagation (baseline)
    II. Genetic Algorithm
    III. Particle Swarm Optimization
- Evaluation metric: Accuracy
- Data split: Train, Validation, Test

2. PERFORMANCE COMPARISON

The following results were obtained:

METHOD	TEST ACCURACY
Backpropagation	1.00
Genetic Algorithm (GA)	0.90
Particle Swarm Optimization (PSO)	0.93

3. CONVERGENCE BEHAVIOR

- Backpropagation converged quickly but issued a convergence warning, indicating sensitivity to iteration limits.
- GA showed gradual improvement due to evolutionary operations.
- PSO demonstrated faster convergence than GA due to social learning and velocity-based updates.

4. STRENGTHS AND LIMITATIONS

Strengths of Metaheuristics:

- Gradient-free optimization
- Better global exploration
- Robust to poor initialization

Limitations:

- Computationally expensive for large networks
- Slower convergence compared to backpropagation
- Risk of overfitting on small datasets

5. KEY INSIGHTS

- Backpropagation achieved the highest accuracy but relied on gradient information.
- PSO provided competitive accuracy with faster convergence than GA.

• GA demonstrated stable performance but required more iterations.
6. IMPROVEMENT OPPORTUNITIES
• Hybrid optimization combining metaheuristics with backpropagation • Cross-validation-based fitness evaluation • Application to larger and more complex datasets
7. CONCLUSION
metaheuristic algorithms are effective alternatives for neural network weight optimization. While traditional backpropagation remains efficient, GA and PSO offer robust and flexible solutions, especially when gradients are unavailable or unreliable.

Grading

The grading will be based solely on the report and viva. Try to make the report representative of your efforts. Marks will be assigned as follows:

1.	Discussion of related Metaheuristic techniques and NN models.	**25%**
2.	Implementation of the algorithms (in Python), analysis, and evaluation of the results of the proposed algorithms.	**50%**
3.	Viva	**25%**

Complex Computing Problem Assessment Rubrics

Criteria and Scales			
Excellent (10-8)	**Good (7-5)**	**Average (4-3)**	**Poor (2-0)**
Criterion 1: Understanding the Problem: How well the problem statement is understood by the student			
Understands the problem clearly and identifies the underlying issues.	Adequately understands the problem and identifies the underlying issues.	Inadequately defines the problem and identifies the underlying issues.	Fails to define the problem adequately and does not identify the underlying issues.
Criterion 2: Research: The amount of research that is used in solving the problem			
Contains all the information needed for solving the problem	Good research, leading to a successful solution	Mediocre research, which may or may not lead to an adequate solution.	No apparent research
Criterion 3: NN and GA Algorithm Design: The completeness of the algorithms			
Algorithms with complete notations	Algorithms with incomplete notations	Algorithms with improper naming conventions and notations	No Algorithms
Criterion 4: Code: How complete and accurate the code is, along with the assumptions.			
Complete the Code according to the Algorithms of the given case with clear assumptions.	Incomplete Code according to the Algorithms of the given case, with clear assumptions	Incomplete Code according to the Algorithms of the given case, with unclear assumptions	Wrong code and naming conventions
Criterion 5: Report: How thorough and well-organized is the solution			
All the necessary information is clearly organized for easy use in solving the problem.	Good information, organized well, that could lead to a good solution.	Mediocre information, which may or may not lead to a solution	No report provided