National University of Computer and Emerging Sciences

Lab Manual
Artificial Intelligence Lab

Instructor: Mariam Nasim

Task: Implementation of Backpropagation in a Multilayer Perceptron (from scratch)

Implement a simple neural network (Multilayer Perceptron) from scratch **without using PyTorch, TensorFlow, or Keras**. The focus is on manually coding **backpropagation** and **gradient updates**.

Dataset: Iris Dataset

- Features: 4 (e.g., sepal length, sepal width, etc.)
- Classes: 3 (Setosa, Versicolor, Virginica)
- Use sklearn.datasets.load_iris() to load

Part 1: Data Preprocessing

1. Load the Iris dataset using sklearn.datasets.
2. Normalize the feature values (you can use sklearn StandardScaler here).
3. One-hot encode the labels (you can use sklearn OneHotEncoder here).
4. Split into training (80%) and testing (20%) sets.

Part 2: MLP Architecture details (implement this yourself. It can be done during forward pass)

- Input layer: 4 neurons
- Hidden layer: 6–10 neurons (ReLU activation)
- Output layer: 3 neurons (Softmax activation)

Part 3: Model Implementation

Forward Pass:

- Compute activations step-by-step.
- Use ReLU for hidden layer, Softmax for output. (implement these activation functions from scratch)

Loss Function:

- Use categorical cross-entropy:

$$L = -\sum y_i \log(\hat{y}_i)$$

Backpropagation:

- Derive and implement:
  - Gradient of loss w.r.t output (Softmax + Cross Entropy combined)
  - Gradient of loss w.r.t hidden weights (ReLU derivative)
  - Weight and bias updates using SGD:

$$W = W - \eta \cdot \frac{\partial L}{\partial W}$$

Implement gradient and loss functions from scratch

Training Loop:

- Train the model for 100 epochs.
- Print training loss every 10 epochs.
- Plot the loss curve

You should output:

- Final weights
- Final accuracy on test set
- Loss values across epochs

Total marks: 20