Assignment 1: Training a Simple Neural Network

Course: Artificial Intelligence / Deep Learning

Duration: 1 week

Topic: Deep Neural Networks

Objective

In this assignment, you will implement a very simple Neural Network (NN) with 3 layers (one neuron per layer) and train it to approximate the following function:

$f(x) = \exp(-\sin(x^2)/2) + x$

Problem Setup

- Input: x_i (scalar)
- Output: $y_i = f(x_i)$
- Neural Network:
  - Layer 1: 1 neuron (w_1, b_1)
  - Layer 2: 1 neuron (w_2, b_2)
  - Layer 3: 1 neuron (w_3, b_3)
- Activation function: Use ReLU or tanh (your choice, but justify).
- Loss function: Mean Squared Error (MSE).
- Optimizer: Gradient Descent.

Data

Generate 25 samples of x uniformly spaced between [-10, 10].
Compute corresponding y = f(x).
Split randomly into:
- Training set: 20 samples
- Test set: 5 samples

Part A: Mathematical Formulation

1. Forward propagation equations:

$a[1] = g(w_1x + b_1)$

$a[2] = g(w_2a[1] + b_2)$

$\hat{y} = g(w_3a[2] + b_3)$

where $g(\cdot)$ is the chosen activation.

2. Loss function:
$J = (1/m)* \Sigma (\hat{y}_i - y_i)^2$

3. Gradient descent update rule:
$w_l \leftarrow w_l - \eta \, \partial J/\partial w_l$
$b_l \leftarrow b_l - \eta \, \partial J/\partial b_l$

where:
- $l = 1, 2, 3$ (layer index)
- η is the learning rate
- $\partial J/\partial w_l$ and $\partial J/\partial b_l$ are gradients

Part B: Implementation

1. Initialize parameters w_l, b_l randomly.
2. Implement forward propagation.
3. Compute the loss J.
4. Implement backward propagation to compute gradients.
5. Update weights and biases using the rule above.
6. Repeat for multiple epochs (e.g., 1000, 10000 etc. iterations).

Part C: Experiments

1. Train the model on the dataset.
2. Plot:
   - The true function f(x).
   - The predictions $\hat{y}$ after training.
3. Report training and test error.

Part D: Analysis

Answer briefly:
1. Did the simple 3-neuron network approximate the function well?
2. How does the choice of activation (ReLU vs tanh) affect performance?
3. Why might deeper/wider networks perform better on complex functions?

Deliverables

- Jupyter Notebook with code implementation on GitHub.
- Plots of true vs predicted function.
- A short PDF write-up (1–2 pages) with answers to Part D.