# Assignment 7: Implementing a Single-Layer Neural Network for Binary Classification

**Objective:** Learn how to design and manually calculate both forward and backward propagation for a single-layer neural network using the sigmoid activation function.

**Instructions:** This assignment engages you in the practical application of neural network fundamentals. You will manually calculate the forward and backward propagation steps and implement the neural network in Python. Submit your work as a single PDF file that includes all required code, calculations, and graphs. Attach supplementary code files (.py and .ipynb) as used in this assignment.

**Step-by-Step Instructions:**

## Question 1: Design and Initialize Neural Network

### 1.1 Network Design and Truth Table Creation

Task: Design a single-layer neural network with four binary inputs (x1, x2, x3, x4) and one binary output. The output should be determined by the logical function: (x1 AND x2) OR (x3 AND x4).

Requirements:

- Create a truth table that lists all possible combinations of inputs and the corresponding binary output.

### 1.2 Manual Forward Propagation Calculation

Task: Perform a manual forward propagation step using an example set of inputs from your truth table. Use the sigmoid activation function. Initialize all weights to 0.5 and the bias to 1.

Requirements:

- Calculate the weighted sum and apply the sigmoid function to obtain the output.
- Show detailed calculations and the formula used for the sigmoid activation.

### 1.3 Manual Backpropagation Calculation

Task: Perform a manual backpropagation step using the output and true label from your truth table to calculate the necessary weight updates. The loss function following:

$$E(\theta) = \left(\mathrm{t} - ((\sum_{i=1}^{m}(\mathrm{x}_i \mathrm{w}_i)) + \mathrm{b})\right)^2$$

Requirements:

- Calculate the error (difference between predicted and actual output).
- Compute the gradient of the loss function with respect to each weight using the chain rule.
- Show detailed step-by-step calculations, including the update of weights using a learning rate of 0.001.
- Provide mathematical formulas for error calculation, gradient computation, and weight update.

**1.4 Implementation and Training in Python**

Task: Implement and train the neural network in Python based on the designs and calculations above. Initialize all weights to 0.5 and the bias to 1, and train the network over 1000 iterations.

Requirements:

- Provide Python code that initializes the network, performs forward propagation, computes the error, adjusts the weights (backpropagation), and iterates this process.
- Plot the evolution of the loss or accuracy over the iterations.
- Discuss the effectiveness of your neural network based on the output from training.

**Submission Requirements:**

- Submit your completed assignment as a PDF containing all necessary code snippets and graphs.
- Attach corresponding .py and .ipynb files containing executable code.
- Ensure all files are clearly labeled and organized.
- Late submissions will be subject to deductions according to the policy.

**Evaluation Criteria:** Your assignment will be graded based on the accuracy of your implementation, the clarity of your presentation, and the completeness of your submission. Specific attention will be given to how well you follow these guidelines:

- **Step-by-Step Explanation:** This assignment requires clear, step-by-step explanations. You must explicitly label and explain each step (e.g., Step 1, Step 2, Step 3, Step 4). Failure to provide detailed explanations for each required step will result in a proportional deduction in your score. For example, if the assignment requires four steps and you complete only three, this will result in a score of 75 out of 100.