Computer-Aided Digital System Design

Fall 2024 - Project
Deadline:

Introduction

In this project, you will design and implement a **Multi-Layer Perceptron (MLP)**, a supervised learning algorithm used for solving classification problems. The goal is to implement the **feed-forward (prediction) phase** of an MLP, which takes an input sample and produces a predicted class (binary classification: 0 or 1). You will be provided with the necessary weights for the network, so this project will focus only on the prediction phase, and you are not required to implement the learning phase (such as Stochastic Gradient Descent or other optimization algorithms).

The activation function used in this project is the **hyperbolic tangent (tanh)**, which you will implement using the **CORDIC algorithm**. Figure 1 provides an overview of MLP architecture, highlighting the structure of input, hidden, and output layers along with their interconnections.

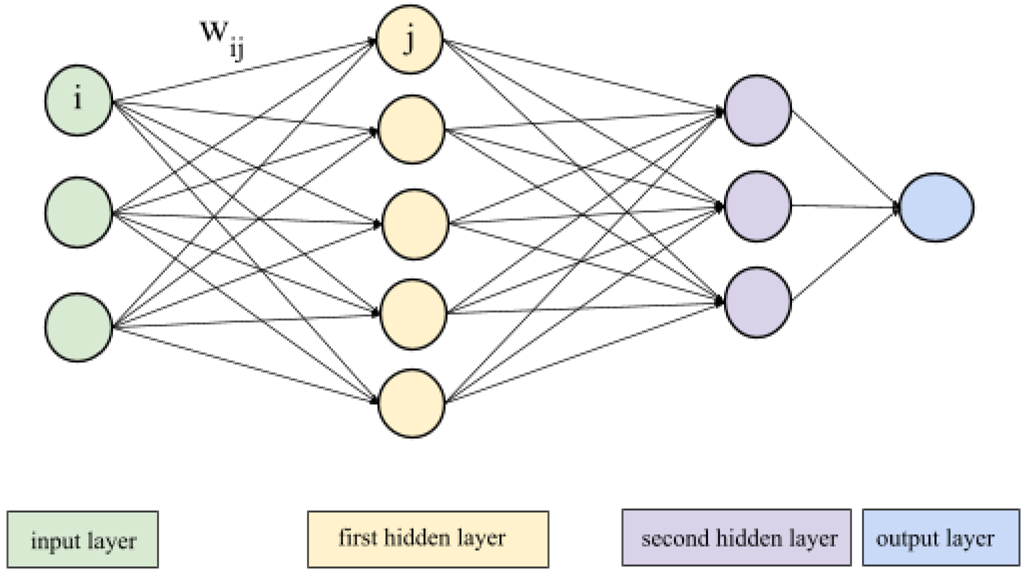

Figure 1: Architecture of an MLP illustrating the input layer, hidden layers, and output layer.

Project Phases

This project is divided into three phases, each building upon the previous one to implement a complete MLP network.

Phase 1: CORDIC Implementation of Tanh

The first step is to implement the **tanh** activation function using the **CORDIC (COordinate Rotation DIgital Computer)** algorithm. CORDIC is an efficient iterative method used to compute hyperbolic and other mathematical functions. In this phase, you will:

- Implement the **tanh** function using CORDIC.
- Choose an appropriate resolution for the algorithm.
- Ensure that the design works correctly for the entire range of input values.

Note: Determine the appropriate number of iterations yourself. Bonus marks are available for finding the optimal resolution for CORDIC; otherwise, use 11 bits for the fractional part.

Phase 2: Single Node Implementation

Once the **tanh** function is implemented, the next step is to design and implement a **single node** of the neural network. A single node performs a **Multiply-Add operation** between the input vector and the weight vector. As shown in Figure 2, the single node performs the operations of multiplying the input vector with the weight vector and passing the result through the activation function (tanh). The steps for this phase are as follows:

- Implement the **dot product** of the input vector X and the weight vector.
- Pass the resulting value through the **tanh** activation function.
- The output of this node will be the input for the next layer.

The weight values will be stored in **RAM**.

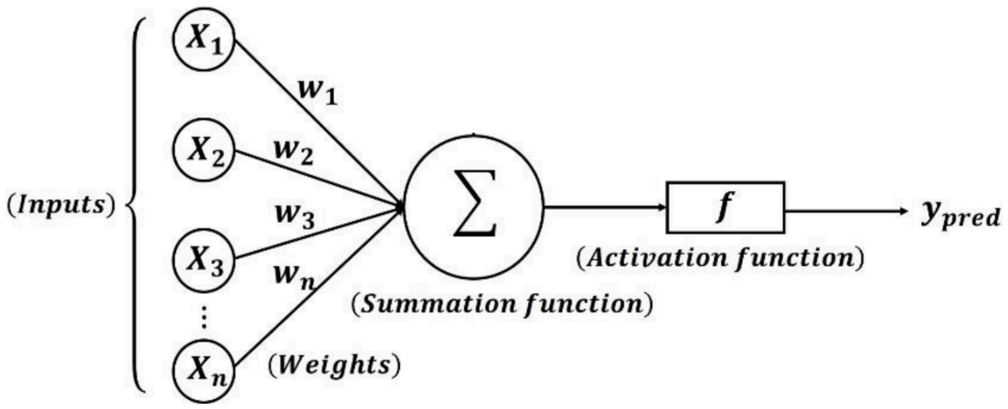

Figure 2: Illustration of a single neuron, including input features, weights, activation function, and output.

Phase 3: Multi-Layer Perceptron (MLP) Implementation

In the final phase, you will connect multiple nodes to form a complete **Multi-Layer Perceptron (MLP)**. The network structure for this project consists of four layers:

- **Input layer:** 3 neurons
- **First hidden layer:** 5 neurons
- **Second hidden layer:** 3 neurons
- **Output layer:** 1 neuron

The input to the network will consist of a 3-element vector, and the output will be a binary classification (0 or 1). You will use the provided initial weights for the connections between layers.

The weights for the connections are as follows:

- **Weights for input to hidden layer 1:**

$$\begin{vmatrix} \mathbf{0.2} & \mathbf{-0.5} & \mathbf{0.8} & \mathbf{-0.3} & \mathbf{0.6} \\ \mathbf{-0.4} & \mathbf{0.7} & \mathbf{-0.1} & \mathbf{0.9} & \mathbf{-0.2} \\ \mathbf{0.1} & \mathbf{-0.6} & \mathbf{0.3} & \mathbf{-0.8} & \mathbf{0.5} \end{vmatrix}$$

- **Weights for hidden layer1 to hidden layer 2:**

$$\begin{vmatrix} 0.3 & -0.7 & 0.4 \\ -0.2 & 0.6 & -0.5 \\ 0.5 & -0.1 & 0.8 \\ -0.6 & 0.2 & -0.9 \\ 0.4 & -0.8 & 0.7 \end{vmatrix}$$

- **Weights for hidden layer 2 to output layer:**

$$\begin{vmatrix} 0.1 \\ -0.3 \\ 0.2 \end{vmatrix}$$

Example Test Cases

1. **Input:** [0.3, 0.1, 0.3]

   Expected Output: 1

2. **Input:** [0.01, 0.2, 0.03]

   Expected Output: 0

3. **Input:** [0.04, 0.9, 0.003]

   Expected Output: 0