Final Project - APCSA

Intro

1. Period 7
2. Aditya Anand, Jason Chao, and Flint Mueller
3. Group Name: CAM
4. Project Title: Neural Networks

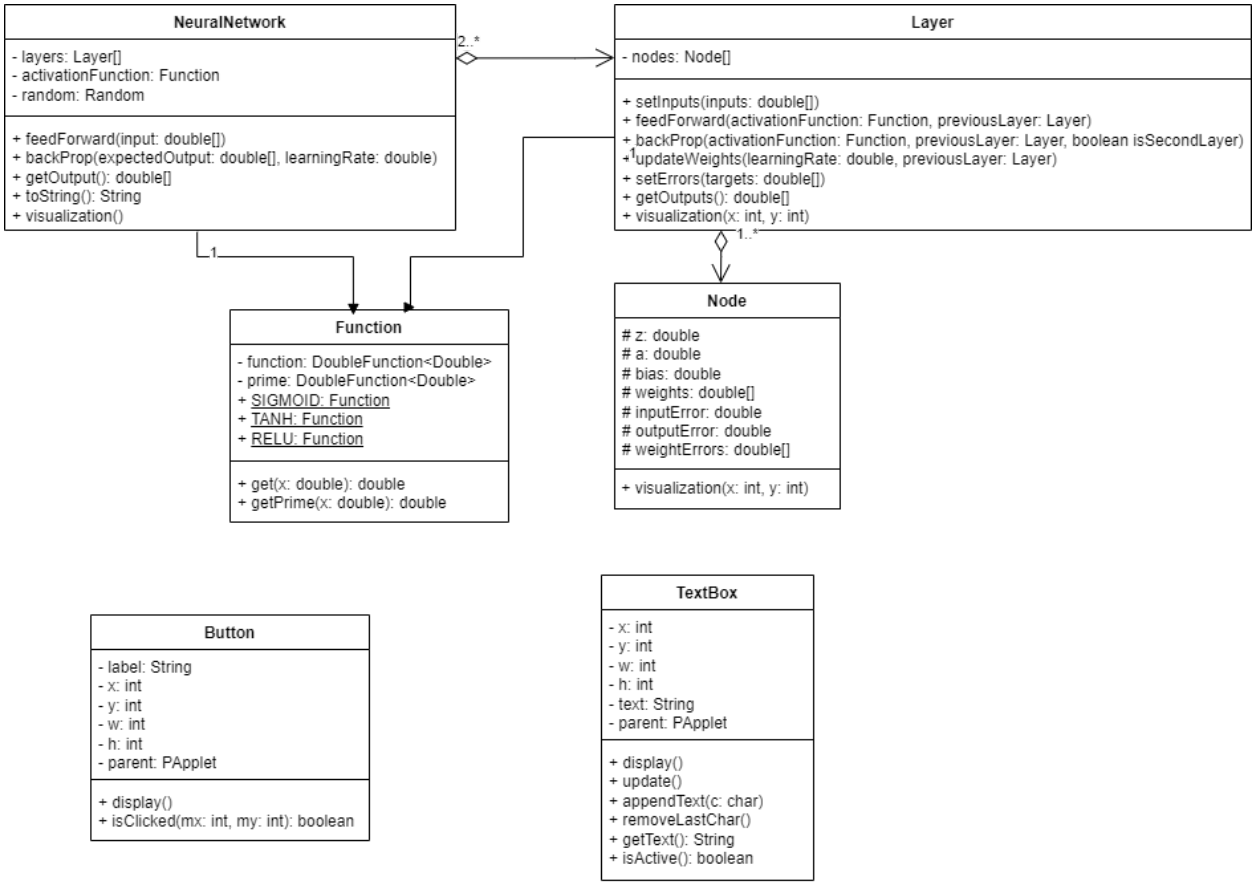
Project Overview

The project is a Neural Network Simulator designed to demonstrate the workings and capabilities of neural networks through various examples. This simulator allows users to understand the mechanics behind neural networks by visualizing and interacting with the different layers, nodes, and the processes of feedforward and backpropagation. There will also be a showcase of different problems that a neural network can solve, and if we have time, an option to test out your own problem.

Functionalities

1. Feedforward Mechanism:
    - Users can input data into the neural network, and the simulator will propagate this data through the network layers to produce an output.
2. Backpropagation Process:
    - After obtaining the output, users can provide the expected output to trigger the backpropagation process, adjusting the weights and biases in the network to minimize the error.
3. Visualization:
    - The simulator provides visual representations of the neural network's architecture, including layers, nodes, and connections, as well as real-time updates during feedforward and backpropagation processes.
4. How it Works:
    - The user-interface will be fairly self-explanatory, and all of the input will either be from buttons or text-boxes.
    - here will be options to change the neural network like adding nodes and layers, and changing the parameters, and each node's weights and bias.
    - There will also be a sidebar with options for each of the sample problems that the neural network can solve.
5. Libraries:
    - No libraries are needed (for now 😈)

UML Diagram



List of Current Functionalities

1. Button Class:
   - Creation of a button with a specified label, position, and dimensions.
   - Displaying the button with appropriate text alignment and color.
   - Detecting if the button is clicked based on mouse coordinates.

2. Function Class:
   - Definitions for common activation functions: Sigmoid, Tanh, and ReLU.
   - Methods to retrieve the function value and its derivative.

3. Node Class:
   - Representation of a node with weights, bias, activations, and error terms.
   - Initialization of weights with random values.

4. Layer Class:
  - Initialization of a layer with a specified number of nodes.
  - Forward propagation through the layer using a specified activation function.
  - Backpropagation through the layer, updating input errors and weight errors.
  - Weight updates based on learning rate and calculated errors.
  - Setting output errors for the output layer.

5. Main Class:
  - Basic setup and drawing of UI components (buttons and text boxes).
  - Handling key presses for input boxes.
  - Parsing input values from text boxes.
  - Handling mouse clicks to trigger feedforward and backpropagation processes.

6. Neural Network Class:
  - Complete implementation of a neural network class that integrates multiple layers and handles overall feedforward and backpropagation processes.

7. UI:
    - UI now shows overall percent accuracy and live updating percent accuracy.

Functionalities Planned for the Next Meeting

1. Make input function work

2. Neural Network Visualization:
  - Develop a method to visualize the neural network structure and activation values in the Main class.

3. UI:
    - Make UI more aesthetically pleasing.

4. Testing:
  - Implement more test cases to ensure the accuracy of the neural network's computations.

Issues Encountered and Solutions

1. Issue: UI Interaction Handling
  - Problem: Inconsistent handling of key presses and mouse clicks for input boxes.

- Solution: Refined the ‘keyPressed’ and ‘mouseClicked’ methods to ensure they correctly identify and modify the active text box.

3. Issue: Activation Function Implementation
   - Problem: Initial confusion on how to properly define and use activation functions and their derivatives.
   - Solution: Created a dedicated `Function` class to encapsulate activation functions and their derivatives, making it easier to manage and use them throughout the code.

4. Issue: Random Initialization of Weights
   - Problem: Determining a suitable method for initializing weights in the `Node` class.
   - Solution: Used a random number generator to initialize weights to values between -1 and 1, ensuring diverse starting conditions for the neural network training.

5. We initially experienced issues with our percent accuracies but have since resolved the issues.

As we address these issues and plan our next steps, the project is progressing well. Future development will focus on enhancing the neural network functionality, **improving the UI**, and ensuring robust testing and validation.