NeuroFuse: Neural Network Framework - Project Report

Vihan Vashishth - IMT2021065 Shishir Shahi - IMT2021029 Allu Pavankarthik Reddy - IMT2021026 Rithik Bansal - IMT2021099

1. Introduction

1.1 Purpose

The purpose of NeuroFuse is to provide a comprehensive neural network framework for developers, data scientists, and machine learning engineers. This document outlines the project specifications and requirements for NeuroFuse version 1.0.

1.2 Intended Audience and Reading Suggestions

This document is crafted for individuals involved in the development, testing, and use of NeuroFuse. It is recommended for both technical and non-technical stakeholders to gain a holistic understanding of the framework.

1.3 Product Scope

NeuroFuse aims to build a powerful and user-friendly neural network framework with a focus on modularity, flexibility, and efficient training.

2. Overall Description

2.1 Product Perspective

NeuroFuse is designed as a self-contained Java library, providing seamless integration into various applications. It operates as both a standalone solution and an embedded component in larger Java frameworks.

2.2 Product Features

Key features of NeuroFuse include layered architecture, weighted connections, activation functions (ReLU, sigmoid), backpropagation, and support for deep learning tasks. It emphasizes user-friendliness and extensibility.

2.3 Operating Environment

NeuroFuse is platform-agnostic, capable of running on any operating system supporting Java (Windows, macOS, Linux). It lacks a graphical user interface and is accessed through a Java programming interface.

2.4 User Interfaces

NeuroFuse is primarily accessed through a console-based interface or script execution. It does not possess a graphical user interface (GUI).

2.5 Hardware Interfaces

The framework does not require specific hardware interfaces and is compatible with standard personal computer hardware.

2.6 Software Interfaces

NeuroFuse operates autonomously without external software interfaces, libraries, or frameworks. It functions as a self-contained Java library.

3. System Features

3.1 Neural Network Structure

- The neural network consists of three layers: input, hidden, and output.
- Specifically designed for binary classification tasks.

3.2 Class Structure

- Organized into classes: NeuralNetwork, Layer, Neuron, StatUtil, TrainingData.
- NeuralNetwork class manages training and execution.
- Layer class represents neuron layers.
- Neuron class encapsulates neuron properties.
- StatUtil class contains utility functions.
- TrainingData class stores training examples.

3.3 Neural Network Operations

- Forward Propagation: Calculates the output of the neural network given input data.
- Backward Propagation: Adjusts neuron weights based on the error between predicted and expected outputs.

3.4 Training Data

- Includes training datasets tailored for specific tasks, such as solving the XOR logic function.

3.5 Training Loop

- Iteratively updates the neural network's weights to minimize the error between predicted and expected outputs.

3.6 Output Display

- Displays the neural network's output before and after training for user observation and evaluation.

4. Nonfunctional Requirements

4.1 Performance Requirements

- The neural network is designed for efficient and fast execution, with optimized training and prediction times.
- Capable of handling a substantial amount of training data efficiently.
- The response time for training and making predictions shall be within acceptable limits.

4.2 Usability

- The software provides clear and user-friendly documentation for users and developers.
- The user interface, although console-based, is intuitive and easy to navigate.
- The system produces informative logs and error messages to aid in troubleshooting and debugging.

4.3 Portability

- The neural network implementation is platform-independent, compatible with major operating systems (e.g., Windows, Linux, macOS).
- The software is deployable on various hardware configurations, including both CPUs and GPUs.

4.4 Reliability

- The software is robust and reliable, with minimal downtime or crashes during training and inference.
- The neural network produces consistent and reproducible results for the same input data.

4.5 Scalability

- The software is designed to scale and adapt to larger and more complex neural network architectures, if needed in the future.

5. Design & Architecture (NeuroFuse)

5.1 Detailed Architecture

- Components and Connectors.
- Training Loop: Forward and Backward Propagation.
- File Reading and Writing.

5.2 Mapping to FRs and NFRs

- Functional Requirements (FRs): Neural Network Training, Data Reading/Writing, Prediction.
- Non-Functional Requirements (NFRs): Performance, Modifiability, Reliability.

5.3 Important Design Decisions

- Activation Functions: ReLU for hidden layers, Softmax for output.

- Loss Function: Mean Squared Error.
- Architecture Style: Sequential layer approach.

5.4 Detailed Design
- Data Structures: Arrays and Matrices.
- Algorithms: Training Algorithm (Stochastic Gradient Descent).

5.5 User Interface Design
- Screen Design: Console-based operation.
- Navigation: Terminal and script-based.

5.6 The Underlying Layered Architecture
- Input Layer, Hidden Layers, Activation Layers, Loss Calculation Layer, Backpropagation Layers, Output Layer.

5.7 Why Layered Architecture?
- Hierarchical Representation.
- Modularity and Abstraction.
- Backpropagation for Training.
- Feature Hierarchy.
- Flexibility and Scalability.

6. Testing Overview

8.1 Testing Approach
- Utilizes JUnit for unit testing and functional testing.
- Ensures comprehensive coverage of NeuroFuse components and functionalities.

8.2 Unit Tests

8.2.1 TrainingDataTester

Validates the correct initialization of TrainingData.

8.2.2 StatUtilTest

Verifies utility functions including random float generation, sigmoid, linear functions, and derivatives.

8.2.3 NeuronTest

Tests neuron constructor variations and the update_weight method.

8.2.4 LayerTest

Ensures the proper initialization of hidden, output, and input layers.

8.3 Functional Tests

8.3.1 TrainingTests

Validates the neural network's ability to learn the XOR logic function.

8.3.2 PredictionTests

Evaluates prediction accuracy on new data after training.

8.4 Conclusion

A robust testing suite ensures the reliability and correctness of NeuroFuse.

6. Conclusion

6.1 Summary of NeuroFuse

NeuroFuse, version 1.0, is a robust and versatile neural network framework developed with a focus on modularity, flexibility, and user-friendliness.

6.2 Achievements and Key Features

- Layered architecture for hierarchical representation.
- Efficient forward and backward propagation.
- Support for ReLU and sigmoid activation functions.
- Robust performance, reliability, and scalability.

6.3 Future Enhancements

- Explore additional activation functions and loss functions.
- Extend support for different neural network architectures.
- Enhance documentation and add more tutorials for users.

This consolidated project report provides an in-depth understanding of NeuroFuse, covering its purpose, features, system requirements, design, and architecture. The framework stands as a comprehensive solution for neural network development, meeting the needs of both novice users and experienced developers.

TERMINAL:

```
Output before training
============
0.6849056
0.70576245
0.67657346
0.6926604
============
Output after training
============
0.0698203
0.9355227
0.93801194
0.059676517
```

INPUT:

```
0, 0
0, 1
1, 0
1, 1
```

OUTPUT: