Machine Learning Assigment

Name-Robin
Roll no:56
Branch -CSE(S7)

Neural Network for Linear Regression: y = 2x + 1

Overview
This project implements a custom neural network framework in C++ designed to learn simple linear regression relationships. The specific model is trained to approximate the function `y = 2x + 1` using a single-neuron architecture with backpropagation.

Project Structure
The implementation consists of multiple header files that together form a complete neural network framework:

Core Network Components
- **`network.h`** - Main neural network class managing layers and connections
- **`neuron.h`** - Individual neuron implementation with forward/backward propagation
- **`model.h`** - Pre-configured model setup for the linear regression task

Data Handling
- **`input.h`**, **`label.h`**, **`output.h`**, **`error.h`** - Data container classes
- **`data_set.h`** - Random data generator for training
- **`training.h`** - Training logic and convergence checking

Mathematical Functions
- **`activation.h`** - Activation functions (sigmoid, ReLU, tanh, leaky ReLU, linear)
- **`derivative.h`** - Derivatives for backpropagation
- **`global_enum.h`** - Activation function enumerations

Supporting Files
- **`structures.h`** - Data structures for network connections
- **`unit_test.cpp`** - Main training program
- **`test.cpp`** - Data generation testing
- **`run_many_times.sh`** - Bash script for multiple training iterations

Key Changes Made to Original Files

Critical Fixes Applied:
1. **`global_enum.h`**: Renamed `tanh` to `tanh_act` to avoid naming conflicts with standard library function
2. **`network.h`**: Added explicit template arguments to `std::pair` calls (`std::pair<bool, double>`) for C++11 compatibility
3. **`activation.h` & `derivative.h`**: Updated switch cases to use `tanh_act` and implemented proper mathematical functions

Model-Specific Modifications for y = 2x + 1:

`model.h`
- Simplified architecture to single neuron with linear activation
- Initialized bias to 0.0
- Set learning rate to 0.01 for optimal convergence
- Configured single input-to-neuron connection with trainable weight

`unit_test.cpp`
- Updated test cases for `y = 2x + 1` function:
  - `{0.0, 1.0}` (2×0 + 1 = 1)
  - `{1.0, 3.0}` (2×1 + 1 = 3)
  - `{2.0, 5.0}` (2×2 + 1 = 5)
  - Additional validation cases

`training.h`
- Enhanced training algorithm with progress monitoring
- Added convergence checking with configurable tolerance (delta = 0.1)
- Implemented step-by-step output for training visualization

`neuron.h`
- Improved backpropagation with proper gradient calculation
- Fixed weight update logic for linear regression
- Added bias adjustment during training

Model Architecture

1.log

Training neural network to learn y = 2x + 1
==

Training with: x = 0, expected y = 1
Input: [0.000000]
Step 100: Output: [0.630270]
Step 200: Output: [0.864667]
Converged in 231 steps
Input: [0.000000]
label: [1.000000]
Output: [0.900895]
L-ID: 0 ,N-ID:0 Bias: [0.901886]   Delta: [0.099105]
In weights: [1.000000]
Out weights: [1.000000]

✓ Training successful for this case

Training with: x = 1, expected y = 3
Input: [1.000000]
Step 100: Output: [2.593995]
Step 200: Output: [2.851389]
Converged in 240 steps
Input: [1.000000]
label: [3.000000]
Output: [2.900583]
L-ID: 0 ,N-ID:0 Bias: [1.901578] Delta: [0.099417]
In weights: [1.000000]
Out weights: [1.000000]

✓ Training successful for this case

Training with: x = 2, expected y = 5
Input: [2.000000]
Step 100: Output: [4.593881]
Step 200: Output: [4.851347]
Converged in 240 steps
Input: [2.000000]
label: [5.000000]
Output: [4.900555]
L-ID: 0 ,N-ID:0 Bias: [2.901550] Delta: [0.099445]
In weights: [1.000000]
Out weights: [1.000000]

✓ Training successful for this case

Training with: x = 3, expected y = 7
Input: [3.000000]
Step 100: Output: [6.593870]
Step 200: Output: [6.851343]
Converged in 240 steps
Input: [3.000000]
label: [7.000000]
Output: [6.900553]
L-ID: 0 ,N-ID:0 Bias: [3.901547] Delta: [0.099447]
In weights: [1.000000]
Out weights: [1.000000]

✓ Training successful for this case

Training with: x = -1, expected y = -1

Input: [-1.000000]
Step 100: Output: [0.442518]
Step 200: Output: [-0.471992]
Step 300: Output: [-0.806732]
Converged in 366 steps
Input: [-1.000000]
label: [-1.000000]
Output: [-0.900440]
L-ID: 0 ,N-ID:0 Bias: [0.098564]   Delta: [-0.099560]
In weights: [1.000000]
Out weights: [1.000000]

✓ Training successful for this case

Training with: x = 10, expected y = 21
Input: [10.000000]
Step 100: Output: [16.969416]
Step 200: Output: [19.524676]
Step 300: Output: [20.459984]
Step 400: Output: [20.802337]
Converged in 468 steps
Input: [10.000000]
label: [21.000000]
Output: [20.900203]
L-ID: 0 ,N-ID:0 Bias: [10.901200]   Delta: [0.099797]
In weights: [1.000000]
Out weights: [1.000000]

✓ Training successful for this case

Final Model Test:
=================
x = -2 -> predicted: 8.9012, expected: -3, error: 11.9012
x = -1 -> predicted: 9.9012, expected: -1, error: 10.9012
x = 0 -> predicted: 10.9012, expected: 1, error: 9.9012
x = 1 -> predicted: 11.9012, expected: 3, error: 8.9012
x = 2 -> predicted: 12.9012, expected: 5, error: 7.9012
x = 3 -> predicted: 13.9012, expected: 7, error: 6.9012
x = 4 -> predicted: 14.9012, expected: 9, error: 5.9012