micrograd++: A 500 line C++ Machine Learning Library

Gautam Sharma
Independent Researcher
gautamsharma2813@gmail.com

***Abstract*—micrograd++ is a pure C++ machine learning library inspired by Andrej Karpathy's Python implementation of micrograd. This header only library aims to provide a simple yet powerful framework for building and training machine learning models. By leveraging the performance efficiency of C++, micrograd++ offers a robust solution for integrating machine learning capabilities directly into C++-based systems and applications.**

I. Introduction

Traditionally, all machine learning libraries are extremely bulky and very hard to integrate as third party dependencies. This aspect scares practitioners to adopt a C++ based machine learning library for prototyping. Python remains the first choice in that front. This library changes that by providing:

- **Modern C++ features**: micrograd++ is written entirely in C++17.
- **Smart pointers**: This library is memory safe.
- **Header only**: No complicated build process. Just include one header and you are good to go.

One of the major goals when developing this library was to make it simple and short. When clobbered into a single file by removing all the comments, micrograd++ comes out to be < 500 lines. When compiled with the highest optimization -O3, the static library comes out to be 122KB.

Currently, micrograd++ is only supported on CPU. This is by design. Most of Machine learning tasks that run on real life devices like embedded devices, phone, etc, do not have access to GPU. To bridge that gap, micrograd++ let's any user train a neural network in C++ and ship that to any edge device.

II. Relevance

The development of micrograd++ is driven by the need for high-performance machine learning libraries that can be seamlessly integrated into C++ environments. Existing machine learning frameworks, predominantly implemented in Python, often face performance limitations when applied to large-scale, real-time applications in C++. micrograd++ addresses these challenges by offering a pure C++ implementation that ensures high performance and efficiency.

Moreover, micrograd++ retains the educational value of the original micrograd library, making it accessible to both beginners and experienced practitioners in the field of machine learning. The library's design simplifies the learning process for those new to machine learning concepts, while also providing the necessary tools for advanced applications.

III. Discussion

A. Technical Description

micrograd++ is built using modern C++ standards (C++17), ensuring compatibility and performance. The library can be used as a header-only library, facilitating easy integration into existing C++ projects. CMake is employed for build configuration and management, streamlining the compilation and linking process.

B. Key Features and Functionalities

- **Neural Networks**: micrograd++ provides comprehensive support for creating and training neural networks. The library includes classes for defining layers and neurons, enabling users to construct complex network architectures.
- **Backpropagation**: The implementation of backpropagation in micrograd++ allows for efficient training of models through gradient descent.
- **Gradient Clipping**: To prevent the issue of exploding gradients, micrograd++ incorporates gradient clipping, ensuring stable training processes.

C. Unique Aspects and Innovations

- **Pure C++ Implementation**: Unlike many existing libraries that rely on Python bindings, micrograd++ is entirely implemented in C++. This not only enhances performance but also allows for seamless integration with C++ applications.
- **Modern C++** : No more dangling references with smart pointers and a similar syntax to popular python libraries.
- **Header only** : micrograd++ comes with *microgradpp.h*, that contains all the headers in one file.
- **Small footprint** : libmicrogradpp.a is only 122KB.
- **Inspiration from micrograd**: Drawing inspiration from the simplicity and educational focus of the original micrograd library, micrograd++ brings these benefits to the C++ ecosystem, making it an ideal choice for both learning and development.

IV. Completion Status

A. Completed Work

The core library of micrograd++ has been fully implemented, encompassing essential classes such as `Value`, `Layer`, and `MLP` (Multi-Layer Perceptron). The library includes examples demonstrating its usage, such as a simple

multi-layer perceptron (MLP) and a computer vision application. Additionally, basic documentation is provided in the form of a README file, guiding users through the setup and usage of the library.

B. Planned Work

Future development plans for micrograd++ include the following:

- **Web Assembly compatibility**: Make micrograpdpp compatible with web assembly to make it work on the web on client side.
- **Optional GPU support**: Make micrograpdpp compatible with modern GPU frameworks.
- **CI/CD Pipeline**: Establishing a continuous integration and continuous deployment (CI/CD) pipeline using GitHub Actions to automate testing and deployment processes.

V. Sample Code

A. Example Usage

The following code snippet demonstrates the basic usage of micrograd++ for performing operations and backpropagation on scalar values.

```
#include <iostream>
#include "Value.hpp"

int main() {
auto a = micrograpdpp::Value::create(2.0);
auto b = micrograpdpp::Value::create(3.0);
auto c = a * b;
c->backProp();

std::cout << a << std::endl;
std::cout << b  << std::endl;

    return 0;
}
```

B. Detailed Description of Core Components

Value Class: The `Value` class represents a single scalar value within the computation graph. It supports basic arithmetic operations with automatic differentiation, enabling efficient gradient computation. The class also implements backpropagation to propagate gradients through the graph.

Layer Class: The `Layer` class represents a single layer in a neural network, composed of multiple neurons. It supports forward propagation and gradient computation, facilitating the construction and training of complex network architectures.

MLP Class: The `MLP` class represents a multi-layer perceptron, composed of multiple layers. It supports the training of models using backpropagation and gradient descent, allowing for the efficient optimization of network parameters.

VI. Supporting Material

A. Results and Performance Metrics

The multi-layer perceptron (MLP) example provided in the library demonstrates successful training, as evidenced by the gradual decrease in the loss function over 50 iterations. This indicates the effectiveness of micrograd++ in training neural networks.

The following piece of code defines a MLP:

```
  /*
* Initialize micrograd
* @input : 3 params
* @layer 1 = 4 neurons
* @layer 2 = 1 neuron -> output
  */
   constexpr double learningRate = 0.0025;

   auto mlp =
   micrograpdpp::MLP(3, {4,1}, learningRate);
```

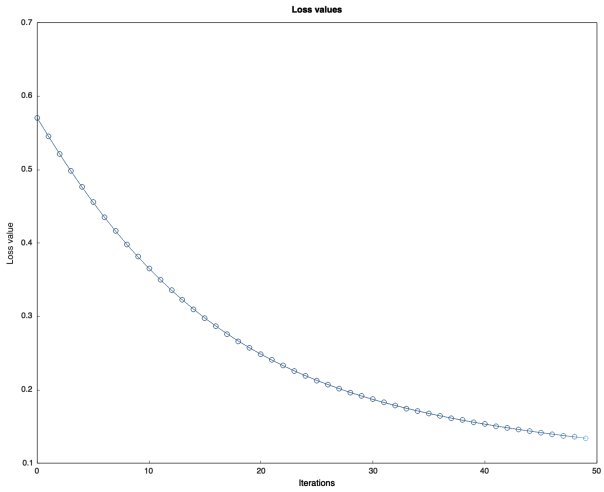

Fig. 1. Loss function of MLP

The computer vision example, which involves learning a German shepherd puppy's face, showcases the library's ability to handle image data and train models to make accurate predictions. The output illustrates the neural network's capability to predict the German shepherd face over time Appendix .

VII. References

- **micrograd**: The original micrograd library by Andrej Karpathy (https://github.com/karpathy/micrograd)

A. Links to Repository and Documentation

- **GitHub**: https://github.com/gautam-sharma1/micrograpdpp
- **Documentation**: Detailed setup and usage instructions are available in the README file of the repository.

VIII. Appendix

Fig. 2. Input to the image model

Fig. 4. Iteration 4

Fig. 3. Iteration 2

Fig. 5. Iteration 6