GSoC 2024 Final Submission

Runtime and Memory Optimizations for Neuronal Networks & KMeans

Objectives

During Google Summer of Code 2024, my goal was to improve the mlpack library by focusing on key algorithm optimizations, with an emphasis on research, testing, and practical implementation.

I concentrated on enhancing the performance of Neural Networks by optimizing convolution operations and implementing post-training quantization to reduce model size and increase speed. For KMeans, I aimed to boost the algorithm's speed and parallelization capabilities using OpenMP, while also restructuring the code to achieve more efficient execution and a smaller binary size. In the case of DBScan, my objective was to accelerate the algorithm through parallelization and reduce its binary footprint by refining space-consuming functions.

Alongside these specific goals, I reviewed and validated existing code examples to ensure they functioned as intended. While SVD optimization was considered, it remained a lower priority compared to the other objectives. Throughout the project, my work was research-driven, with a strong focus on testing various approaches to identify and implement the most effective solutions for improving the mlpack library.

Deliverables

Mlpack examples

https://github.com/mlpack/examples/pull/226 (Merged)

In this contribution, I updated and fixed the Python notebook examples in the mlpack repository. This involved aligning the notebooks with the latest mlpack API changes, correcting typos and resolving bugs. I ensured all notebooks are fully functional by testing and making necessary adjustments. Additionally, I added a new notebook for contact tracing using DBSCAN, adapted from the C++ version. To address SSL verification issues, I updated dataset URLs to a more reliable source, eliminating certificate errors and ensuring smooth operation. These updates improve the usability and reliability of the Python examples in the repository.

Optimization Backward function (Convolution)

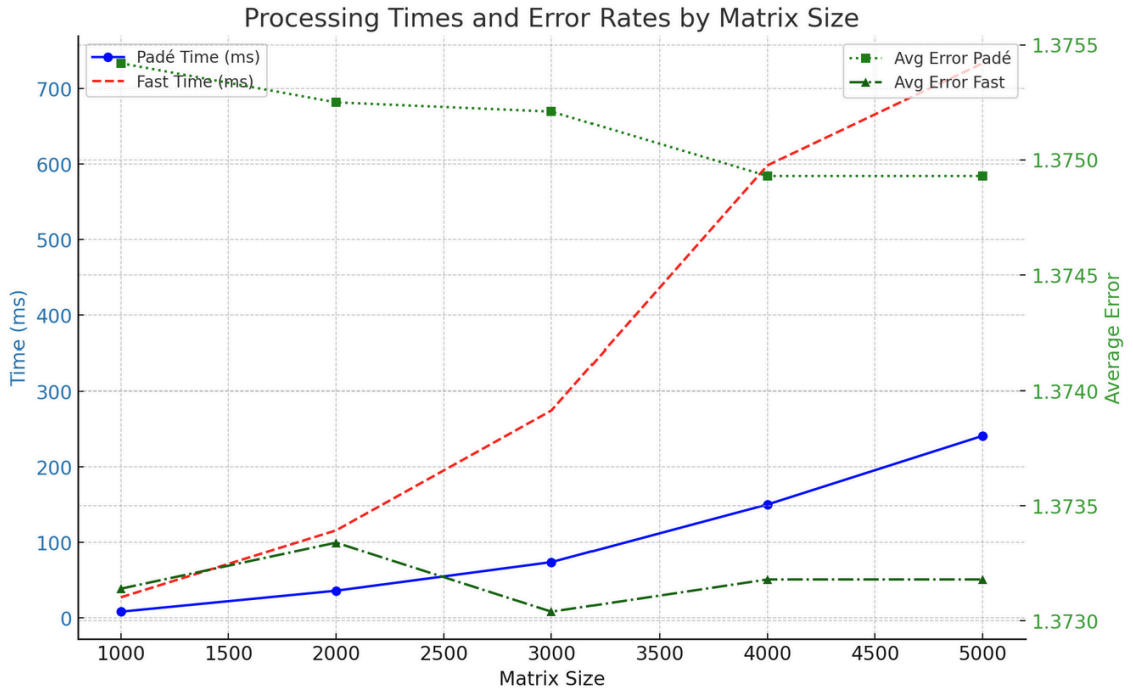
https://github.com/mlpack/mlpack/pull/3738 (Merged)

In this contribution, I optimized the backward function in the convolution code, focusing on padding, rotation, and dilation operations to reduce computational overhead and improve performance. Padding values were integrated directly into the convolution operation, and the filter rotation was replaced with matrix flipping or transposing combined with reversing, embedded within the convolution step. Dilation factors were also embedded directly into the convolution. These changes resulted in a speedup, with the optimized code reducing the time from 73222ms to 50517ms in benchmarks. The changes were tested, adjusted, and successfully merged.

Optimization Dropout and Logsoftmax

https://github.com/mlpack/mlpack/issues/3662 (Closed)
https://github.com/mlpack/mlpack/pull/3684 (Merged)
https://github.com/mlpack/mlpack/pull/3685 (Merged)

I worked on optimizing the LogSoftmax and Dropout layers in mlpack to replace the transform function, ensuring compatibility with the Bandicoot library.



For the LogSoftmax layer, I tested the Padé Approximant as an alternative to the Fast Approximation method for exp(-x). Although the Padé method improved accuracy for smaller X values, it caused performance issues with larger values typical in datasets

like MNIST. After benchmarking, the OpenMP-optimized Fast Approximation was chosen for its better balance of speed and accuracy, leading to faster training times.

In the Dropout layer, I implemented a "Find and Fill" method to replace the transform function by directly manipulating matrix elements. While this method was slightly slower on larger matrices, it maintained accuracy and, with OpenMP, provided a 23.2% speedup during MNIST training.
Both layers now support dual implementations for Bandicoot and Armadillo, enabling GPU compatibility.

Benchmark repository
https://github.com/MarkFischinger/mlpack-benchmarks

I conducted research on RDTSC to explore its potential for optimal testing. While it showed some initial promise, further investigation revealed it wasn't the best fit for our needs. Consequently, I developed a default framework for testing, which was critical for maintaining the accuracy of our benchmarks. This foundational work was a top priority, as it directly impacted the reliability of all subsequent benchmarks. The repository successfully established a solid benchmarking strategy.

DBSCAN
https://github.com/mlpack/mlpack/pull/3771
(Passes all tests apart from a server-side error)

I optimized the DBSCAN implementation in mlpack to improve performance, particularly for larger datasets.

Key improvements included parallelizing the centroid calculation and cluster assignments using OpenMP, which significantly sped up these operations. I also optimized batch clustering and introduced atomic operations to minimize bottlenecks in multi-threaded sections. Additionally, I improved memory locality to boost cache efficiency.

These changes resulted in a noticeable speedup in DBSCAN, especially during processing of large datasets. The implementation was tested and refined based on feedback, ensuring it is both efficient and scalable.

Means Optimizations (Naive, Hamerly, Elkan)

- **Naive K-Means**: PR #3762 (Main, has verified improvements, but needs further investigation)

- **Hamerly K-Means**: PR #3761 (Passes all tests apart from a server-side error)
- **Elkan K-Means**: PR #3764 (Passes all tests)

Naive K-Means

OpenMP was applied to the Naive K-Means algorithm, achieving a 6.3% reduction in execution time. After experimenting with fixed block sizes, I switched to a dynamic segmenting strategy based on available threads, improving efficiency and memory access patterns.

Elkan K-Means

In the Elkan K-Means implementation, OpenMP optimization led to a 23.6% speedup. The key changes involved parallelizing distance calculations and centroid updates while restructuring critical sections to minimize overhead and enhance parallel execution.

Hamerly K-Means

The Hamerly K-Means saw the most significant improvement, with a 42.6% reduction in execution time. The optimization focused on parallelizing distance computations and efficiently managing thread-safe operations using OpenMP reduction and atomic operations.

Post-training Quantization

https://github.com/mlpack/mlpack/pull/3750

I implemented key aspects of post-training quantization for neural networks, converting model weights from float32 to int8. The design is flexible, allowing custom quantization strategies, with examples and documentation provided. Following Ryan's feedback, I refactored the quantization API to ensure a clearer and more user-friendly interface. The updated implementation now needs to pass the newly created tests.

To-Do: SVD Benchmarks

Benchmark the current SVD methods against arma::svd() in Armadillo to evaluate their performance. This task will help determine if any optimization is needed.

Acknowledgement

I would like to express my sincere gratitude to Ryan and Omar for their guidance and support throughout this project. Your mentorship has been crucial in helping me overcome challenges and gain a better understanding of the work. I truly appreciate the time and effort you both invested in helping me succeed. Thank you for being exceptional mentors.