Benchmark Specifications

This document details the benchmarks that have been written, to evaluate the performance requirement. It is intended to be a reference document for markers which is more readable than the actual benchmark code whilst also providing some explanation for the rationale behind the methodology.

The Benchmarking Framework

A custom benchmarking suite was written for the project in order to avoid having to include another dependency and to give more granular control over the benchmarking methodology. Scaffolding code is located in ./bench/bench_helpers.hpp. A benchmark is run by calling benchmarkRunner, which is passed a pointer to a BenchFixture object which defines the body of the benchmark via its two methods setup and run. The runner does 10 warmup runs, followed by 50 proper runs. The mean execution time, standard deviation, min, and max are calculated and outputted. The benchmark fixtures are initialized in ./bench/layer_bench.cpp, ./bench/utils_bench.cpp, and ./bench/system_bench.cpp.

Utils Benchmarks

Benchmarks are located in ./bench/utils_bench.cpp. The benchmarks test the performance of individual operators defined in utils. They are expressed as child classes of the abstract BenchFixture class. BenchFixture has two virtual methods:

- setup() for any initialization/precursor tasks needed to get internal state ready for benchmarking, but not intended to be included within the measured execution time
- run() contains the code for which we actually want to measure the execution time

Benchmark data sizes were chosen to cover a decent range of the sorts of matrix shapes one could expect to see in a typical neural network architecture. The actual data used is randomly generated at runtime.

Benchmark Name	Description of Benchmark Data	Functionality Benchmarked
matrix_mul_100x100_100x100	100x100 matrix a 100x100 matrix b	Matrix-matrix multiplication
matrix_mul_500x500_500x500	500x500 matrix a 500x500 matrix b	As above
matrix_mul_1000x1000_1000x1000	1000x1000 matrix a 1000x1000 matrix b	As above
matrix_div_100x100_100x100	100x100 matrix a 100x100 matrix b	Matrix-matrix division
matrix_div_500x500_500x500	500x500 matrix a 500x500 matrix b	As above
matrix_div_1000x1000_1000x1000	1000x1000 matrix a 1000x1000 matrix b	As above
matrix_transpose_100x100	100x100 matrix	Matrix transposition

matrix_transpose_500x500	500x500 matrix	As above
matrix_transpose_1000x1000	1000x1000 matrix	As above
matrix_mean_100x100	100x100 matrix	Taking mean of all values in a matrix
matrix_mean_500x500	500x500 matrix	As above
matrix_mean_1000x1000	1000x1000 matrix	As above

Evaluation

I believe that these performance tests are suitable and comprehensive. The 10 warm up runs help prevent anomalous runtimes from the first few iterations from skewing the results, and taking the mean of the 50 proper runs gives a reliable average performance indication. With more time, this approach could be extended to all of the functions defined in utils, giving a good overview of performance for this component. One improvement could be to collect more metrics during execution – for example, reporting memory usage or CPU temperature. These factors can significantly constrain performance so it would be helpful to log them. Additionally, future work could integrate competitor libraries into the benchmarking framework. We could then run the same benchmarks across many different libraries and compare our library's performance against the competition.

Layer Benchmarks

Benchmarks are located in ./bench/layer_bench.cpp. The methodology is much the same as for the utils benchmarks. Here we test the speed of backward and forward propagation at different input sizes and for different types of layers.

Benchmark Name	**Description of Benchmark Data**	**Functionality Benchmarked**
dense_fprop_100x100	100x100 matrix	Dense layer forward propagation
dense_fprop_500x500	500x500 matrix	As above
dense_backprop_100x100	100x100 matrix	Dense layer backward propagation
dense_backprop_500x500	500x500 matrix	As above
dropout_fprop_100x100	100x100 matrix	Dropout layer forward propagation
dropout_fprop_500x500	500x500 matrix	As above
dropout_backprop_100x100	100x100 matrix	Dropout layer backward propagation
dropout_backprop_500x500	500x500 matrix	As above

Evaluation

Much the same as with the evaluation for the utils benchmarks (above), I believe these performance tests are adequate, but could be enhanced by logging more statistics to develop a more detailed understanding of which factors are constraining performance.

System Benchmarks

The system benchmark measures the performance of a single epoch of a full neural network model for classifying items of clothing. The model architecture is described in the Test Specification document. Although it follows the same architecture, it is trained on random data rather than actual training images.

Evaluation

I believe this is a good system-level benchmark, except for the major shortcoming that the model used doesn't incorporate all of the components implemented in the library (e.g., no dropout layers). Future work would expand on this by adding more complex models for system testing which use the full breadth of features offered by the library.