Test and Benchmark Results

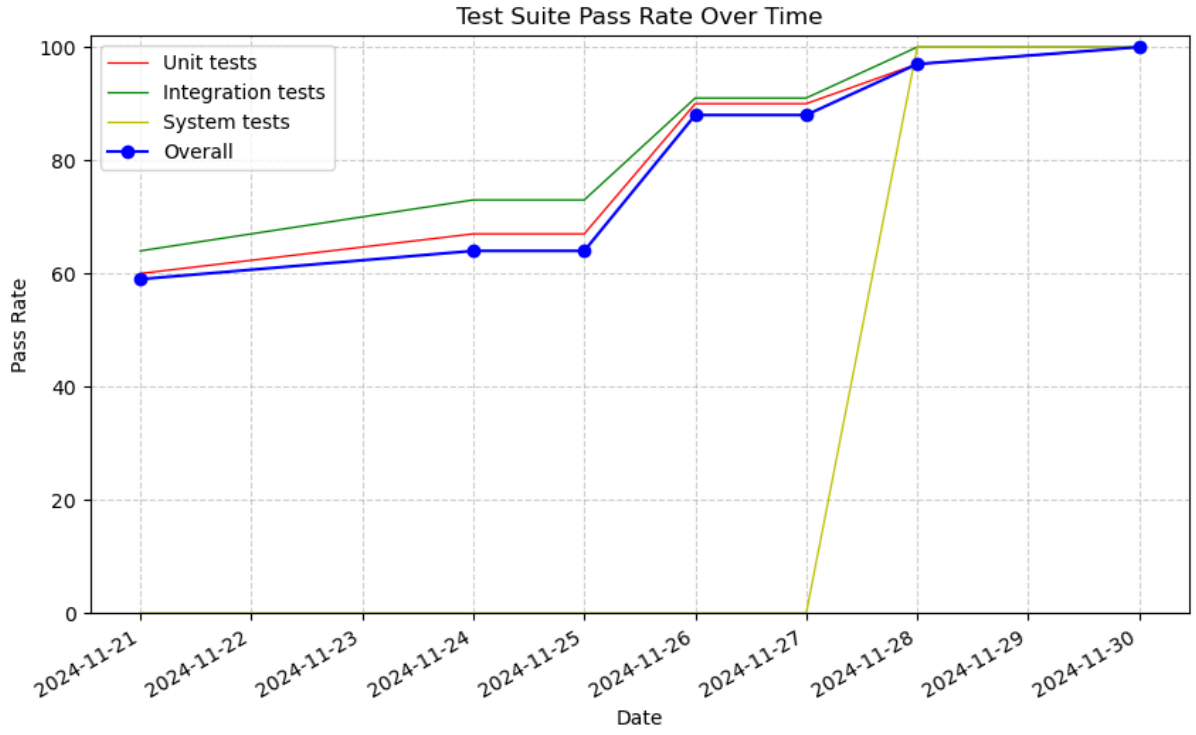
Test Results

Target Coverage Levels

We aim to achieve comprehensive code coverage levels to ensure the system meets its functional requirements. Unit tests aim for a minimum line coverage of 90% for the file being tested. For branch coverage we set a lower minimum threshold of 75%. Integration testing should focus on achieving 100% line coverage. Ideally, system testing should verify 100% of the functional requirements defined in the requirements document.

Statistical Analysis

Overall, 62 tests were developed for this coursework. 20 of these were performance tests, and 42 were testing functional requirements. Of those 42, 30 were unit tests, 11 were integration tests, and 1 was an overall system test.
Developing and implementing the test suite uncovered a number of errors in the library code which had to be resolved. The figure below shows how the test pass rate evolved over time.



We can observe a strong correlation between unit test pass rate and integration test pass rate, indicating that improvements in unit test pass rate lead to improvements in integration test pass rate. Similarly, one most of the unit and integration tests pass, so does the system test. The system test first passes on 2024-11-28, but one unit test is still failing, suggesting that there is a gap in the system test's coverage which does not cover the case being tested by the one unit case that was still failing.
Together, the unit and integration tests for utils.cpp achieve a line coverage of 100% and a branch coverage of 74.7%, meeting our target coverage levels outlined above (open ./coverage/coverage.html to view detailed coverage reports).

The unit and integration tests for Layer.cpp achieve a line coverage of 81.9% and a branch coverage of 63.2%. Inspecting the gcovr output for Layer.cpp reveals that missed lines are caused by esoteric edge cases which aren't triggered by the test code, as well as some setter functions which weren't tested in the interests of timekeeping. These levels are below our targets, but it would be trivial to get them higher.
The integration tests for Layer.cpp also, by proxy, execute 38.2% of lines in Activations.cpp, and 42.1% of the lines in Loss.cpp, despite there not being explicit tests for those files.

Deficiencies and How They Might be Resolved

Deficiency: Gaps in the testing process. We currently only have one system level test which doesn't incorporate all of the features of the library (e.g., the model tested doesn't use dropout layers). Furthermore, the unit and integration tests for Layer.cpp don't achieve the target of 100% line coverage.
Resolution: Develop more tests. The test suite desperately needs more system tests. Furthermore, we need test cases to capture more esoteric edge cases, such as those missed by the current crop of unit tests for Layer.cpp. This can be done by using the Gcovr test coverage reports to identify which areas of the code aren't being executed by the test suite, and tailor-making test cases to hit those blackspots.

Deficiency: There is more scope for automation. Currently, test outputs are checked against reference outputs which are either manually calculated or computed by trusted external libraries. We could accelerate the test development process by automating test generation.
Resolution: The architecture for such an automation would involve randomly generating test input data, executing the operation we are testing using the trusted external library, capturing its output, then executing the operation using our library and comparing the outputs. This way, we remove the need for hard-coded inputs and outputs in our test cases and add a degree of stochasticity to our testing which should help catch edge cases. With this implemented, the test developer no longer has to worry about manually generating input data and calculating the correct output – they only need to specify the operation being tested and the range of input values.

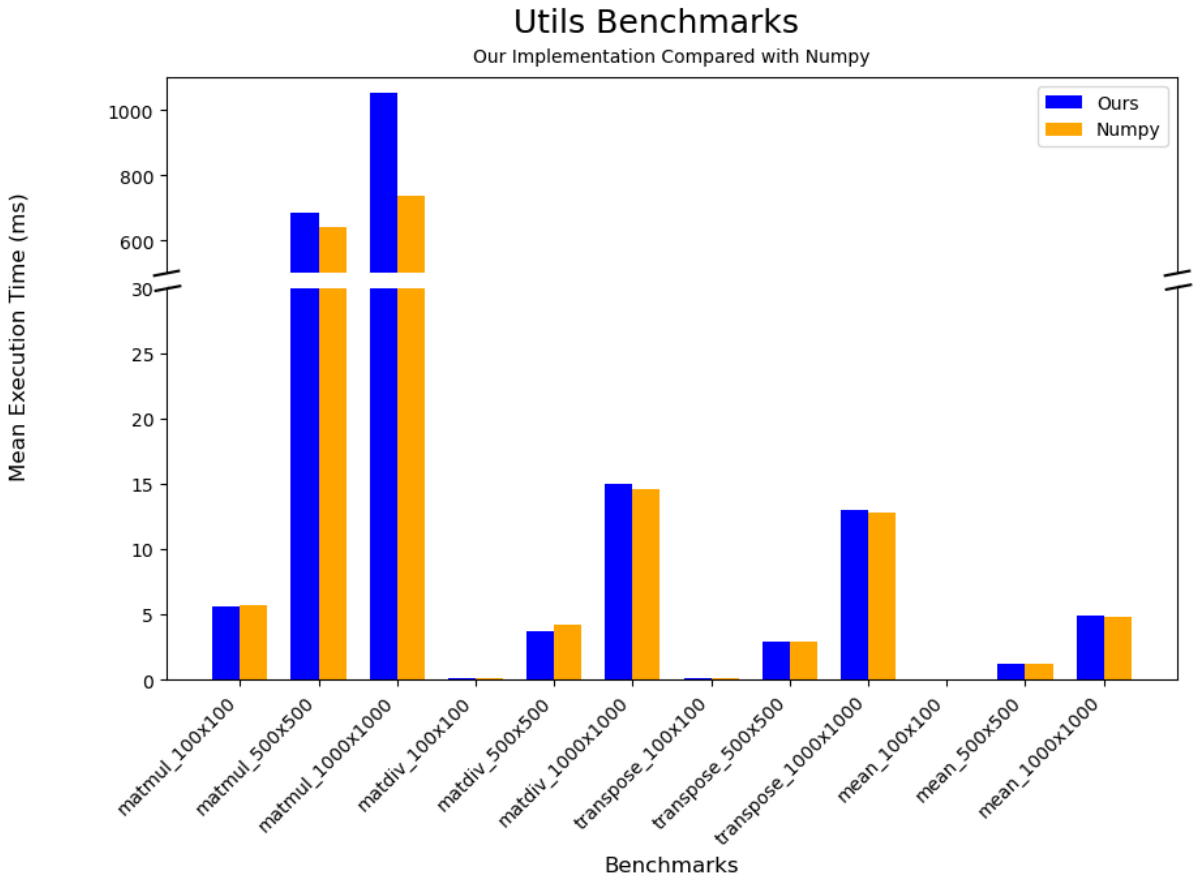
Benchmark Results

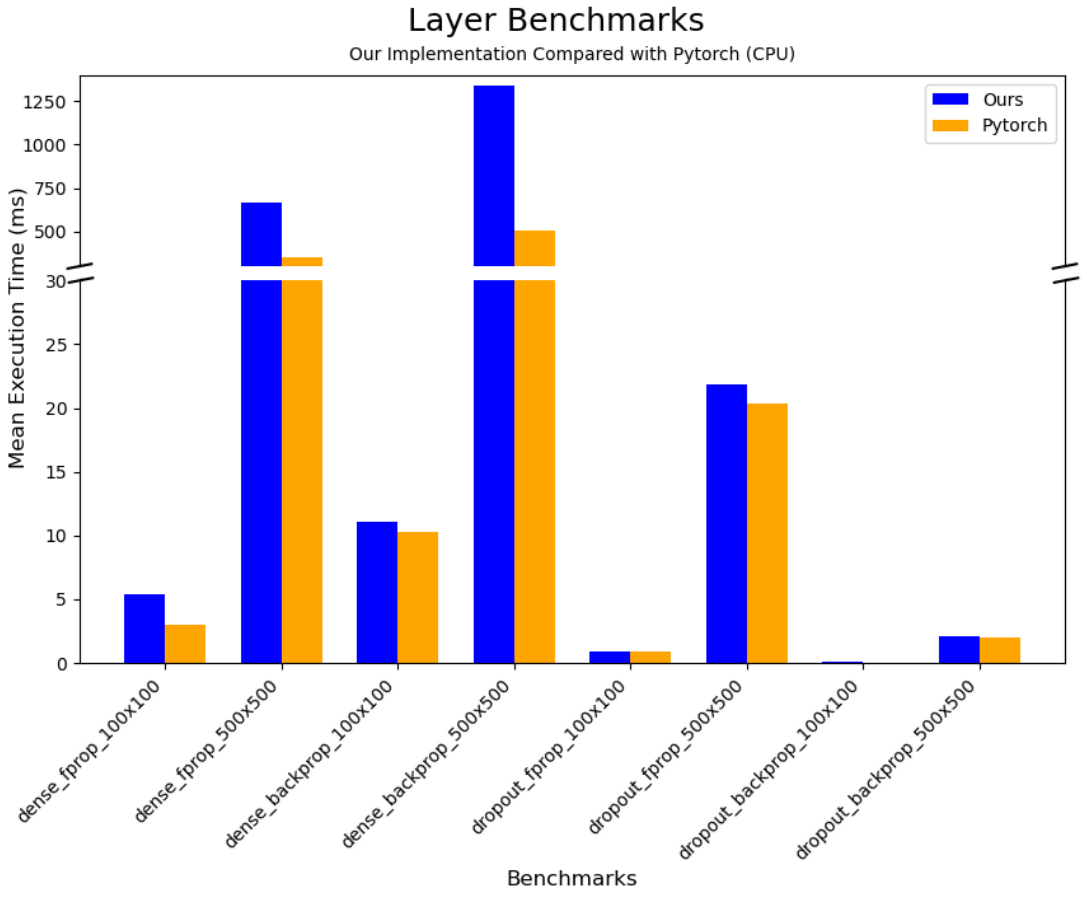
Target Performance Levels

Ideally, we would expect performance to match (or even exceed) that of the current most widely-used libraries in neural networking. To temper expectations, it must be noted that these libraries are enormous projects with many contributors and high budgets, whereas my library is a single-developer endeavour. Therefore, we can set a modest goal of achieving at least 75% of the performance of other libraries in all performance tests.

Statistical Analysis

The graphs below plot the results of performance testing for Layer.cpp and utils.cpp. Performance is measured against the performance of NumPy in the case of utils.cpp, and Pytorch in the case of Layer.cpp. All benchmarking was done on an M1 Mac with GPU acceleration disabled.


Utils Benchmarks
Our Implementation Compared with Numpy
Ours
Numpy
Mean Execution Time (ms)
1000
800
600
30
25
20
15
10
5
0
matmul_100x100
matmul_500x500
matmul_1000x1000
matdiv_100x100
matdiv_500x500
matdiv_1000x1000
transpose_100x100
transpose_500x500
transpose_1000x1000
mean_100x100
mean_500x500
mean_1000x1000
Benchmarks


Layer Benchmarks
Our Implementation Compared with Pytorch (CPU)
Ours
Pytorch
Mean Execution Time (ms)
1250
1000
750
500
30
25
20
15
10
5
0
dense_fprop_100x100
dense_fprop_500x500
dense_backprop_100x100
dense_backprop_500x500
dropout_fprop_100x100
dropout_fprop_500x500
dropout_backprop_100x100
dropout_backprop_500x500
Benchmarks

For utils we achieve our target of 75% performance for all but the 1000x1000 matrix multiplication benchmark. For the Layer benchmarks, we achieve our performance target in all but the 500x500 dense backpropagation. In both cases, we could attempt to hit our performance target by running the code through a profiler, identifying performance bottlenecks, and optimising accordingly. It is likely that further optimising matrix multiplication will have the knock-on effect of improving performance on the Layer benchmarks, because all the functions tested make heavy use of matrix multiplication. This is an important insight because it strongly suggests that any further development effort should focus on optimizing performance of core operations define in utils because they are reused throughout the project and thus give the highest return on investment if optimized.

Deficiencies and How They Might by Resolved

Deficiency: Currently performance testing has only been carried out on 1 hardware platform (M1 Mac). To get a broader understanding of how well the software performs, it needs to be tested on a wide variety of different hardware platforms. Variables we could vary would include amount of memory, or type of CPU.
Resolution: If I had the budget, I would rent virtual testing rigs with different hardware characteristics to run performance testing on. I could then aggregate results and perform a statistical analysis to examine the relationship between the library's performance and the hardware it is running on. A cheaper option could be to use software to simulate changes in hardware; for example one can use control groups in Linux to limit the amount of memory available to a process.

Deficiency: Over-reliance on synthetic data for performance testing. The current performance tests uses synthetically generated data which doesn't accurately reflect the . Usually, the deeper you go in a neural network the more finely-tuned weights become and thus they require more decimal places to accurately represent; in cases like these the level of precision really affects performance, and the current benchmark data doesn't capture that.
Resolution: Use the code from the system tests to load image data from the MNIST fashion dataset and run the benchmarks with that. The code and infrastructure for this is already written as part of the system-level tests, so making this change would simply be a case of changing the code populating the benchmark inputs. Using the MNIST data instead of synthetic data better reflects the kind of data actually used in real-world scenarios.

Test Output Log

```
[==========] Running 83 tests from 9 test suites.
[----------] Global test environment set-up.
[----------] 1 test from HelloTest
[ RUN      ] HelloTest.BasicAssertions
[       OK ] HelloTest.BasicAssertions (0 ms)
[----------] 1 test from HelloTest (0 ms total)

[----------] 20 tests from UtilsUnitTests/0, where TypeParam = float
[ RUN      ] UtilsUnitTests/0.MatMulSmall
```

```
[       OK ] UtilsUnitTests/0.MatMulSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.MatMulMed
[       OK ] UtilsUnitTests/0.MatMulMed (0 ms)
[ RUN      ] UtilsUnitTests/0.MatMulBig
[       OK ] UtilsUnitTests/0.MatMulBig (1 ms)
[ RUN      ] UtilsUnitTests/0.ScalarMulSmall
[       OK ] UtilsUnitTests/0.ScalarMulSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.ScalarMulMed
[       OK ] UtilsUnitTests/0.ScalarMulMed (0 ms)
[ RUN      ] UtilsUnitTests/0.ScalarMulBig
[       OK ] UtilsUnitTests/0.ScalarMulBig (0 ms)
[ RUN      ] UtilsUnitTests/0.MatDivSmall
[       OK ] UtilsUnitTests/0.MatDivSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.MatDivMed
[       OK ] UtilsUnitTests/0.MatDivMed (0 ms)
[ RUN      ] UtilsUnitTests/0.MatDivBig
[       OK ] UtilsUnitTests/0.MatDivBig (0 ms)
[ RUN      ] UtilsUnitTests/0.PowerSmall
[       OK ] UtilsUnitTests/0.PowerSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.ScalarDivSmall
[       OK ] UtilsUnitTests/0.ScalarDivSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.ScalarSubSmall
[       OK ] UtilsUnitTests/0.ScalarSubSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.PowerMed
[       OK ] UtilsUnitTests/0.PowerMed (0 ms)
[ RUN      ] UtilsUnitTests/0.PowerBig
[       OK ] UtilsUnitTests/0.PowerBig (0 ms)
[ RUN      ] UtilsUnitTests/0.TransposeSmall
[       OK ] UtilsUnitTests/0.TransposeSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.TransposeMed
[       OK ] UtilsUnitTests/0.TransposeMed (0 ms)
[ RUN      ] UtilsUnitTests/0.TransposeBig
[       OK ] UtilsUnitTests/0.TransposeBig (0 ms)
[ RUN      ] UtilsUnitTests/0.MeanSmall
[       OK ] UtilsUnitTests/0.MeanSmall (0 ms)
[ RUN      ] UtilsUnitTests/0.MeanMed
[       OK ] UtilsUnitTests/0.MeanMed (0 ms)
[ RUN      ] UtilsUnitTests/0.MeanBig
[       OK ] UtilsUnitTests/0.MeanBig (0 ms)
[----------] 20 tests from UtilsUnitTests/0 (4 ms total)

[----------] 20 tests from UtilsUnitTests/1, where TypeParam = double
[ RUN      ] UtilsUnitTests/1.MatMulSmall
[       OK ] UtilsUnitTests/1.MatMulSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.MatMulMed
```

```
[       OK ] UtilsUnitTests/1.MatMulMed (0 ms)
[ RUN      ] UtilsUnitTests/1.MatMulBig
[       OK ] UtilsUnitTests/1.MatMulBig (1 ms)
[ RUN      ] UtilsUnitTests/1.ScalarMulSmall
[       OK ] UtilsUnitTests/1.ScalarMulSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.ScalarMulMed
[       OK ] UtilsUnitTests/1.ScalarMulMed (0 ms)
[ RUN      ] UtilsUnitTests/1.ScalarMulBig
[       OK ] UtilsUnitTests/1.ScalarMulBig (0 ms)
[ RUN      ] UtilsUnitTests/1.MatDivSmall
[       OK ] UtilsUnitTests/1.MatDivSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.MatDivMed
[       OK ] UtilsUnitTests/1.MatDivMed (0 ms)
[ RUN      ] UtilsUnitTests/1.MatDivBig
[       OK ] UtilsUnitTests/1.MatDivBig (0 ms)
[ RUN      ] UtilsUnitTests/1.PowerSmall
[       OK ] UtilsUnitTests/1.PowerSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.ScalarDivSmall
[       OK ] UtilsUnitTests/1.ScalarDivSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.ScalarSubSmall
[       OK ] UtilsUnitTests/1.ScalarSubSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.PowerMed
[       OK ] UtilsUnitTests/1.PowerMed (0 ms)
[ RUN      ] UtilsUnitTests/1.PowerBig
[       OK ] UtilsUnitTests/1.PowerBig (0 ms)
[ RUN      ] UtilsUnitTests/1.TransposeSmall
[       OK ] UtilsUnitTests/1.TransposeSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.TransposeMed
[       OK ] UtilsUnitTests/1.TransposeMed (0 ms)
[ RUN      ] UtilsUnitTests/1.TransposeBig
[       OK ] UtilsUnitTests/1.TransposeBig (0 ms)
[ RUN      ] UtilsUnitTests/1.MeanSmall
[       OK ] UtilsUnitTests/1.MeanSmall (0 ms)
[ RUN      ] UtilsUnitTests/1.MeanMed
[       OK ] UtilsUnitTests/1.MeanMed (0 ms)
[ RUN      ] UtilsUnitTests/1.MeanBig
[       OK ] UtilsUnitTests/1.MeanBig (0 ms)
[----------] 20 tests from UtilsUnitTests/1 (3 ms total)

[----------] 10 tests from LayerUnitTests/0, where TypeParam = float
[ RUN      ] LayerUnitTests/0.DenseLayerInit
[       OK ] LayerUnitTests/0.DenseLayerInit (15 ms)
[ RUN      ] LayerUnitTests/0.DenseLayerCompute
[       OK ] LayerUnitTests/0.DenseLayerCompute (2 ms)
[ RUN      ] LayerUnitTests/0.DenseLayerBackpropNoReg
```

```
[       OK ] LayerUnitTests/0.DenseLayerBackpropNoReg (0 ms)
[ RUN      ] LayerUnitTests/0.DenseLayerBackpropL1Reg
[       OK ] LayerUnitTests/0.DenseLayerBackpropL1Reg (0 ms)
[ RUN      ] LayerUnitTests/0.DenseLayerBackpropL2Reg
[       OK ] LayerUnitTests/0.DenseLayerBackpropL2Reg (0 ms)
[ RUN      ] LayerUnitTests/0.DropoutLayerComputeTrain
[       OK ] LayerUnitTests/0.DropoutLayerComputeTrain (0 ms)
[ RUN      ] LayerUnitTests/0.DropoutLayerComputeEval
[       OK ] LayerUnitTests/0.DropoutLayerComputeEval (0 ms)
[ RUN      ] LayerUnitTests/0.DropoutLayerBackprop
[       OK ] LayerUnitTests/0.DropoutLayerBackprop (0 ms)
[ RUN      ] LayerUnitTests/0.InputLayerCompute
[       OK ] LayerUnitTests/0.InputLayerCompute (0 ms)
[ RUN      ] LayerUnitTests/0.InputLayerBackprop
[       OK ] LayerUnitTests/0.InputLayerBackprop (0 ms)
[----------] 10 tests from LayerUnitTests/0 (21 ms total)

[----------] 10 tests from LayerUnitTests/1, where TypeParam = double
[ RUN      ] LayerUnitTests/1.DenseLayerInit
[       OK ] LayerUnitTests/1.DenseLayerInit (15 ms)
[ RUN      ] LayerUnitTests/1.DenseLayerCompute
[       OK ] LayerUnitTests/1.DenseLayerCompute (1 ms)
[ RUN      ] LayerUnitTests/1.DenseLayerBackpropNoReg
[       OK ] LayerUnitTests/1.DenseLayerBackpropNoReg (0 ms)
[ RUN      ] LayerUnitTests/1.DenseLayerBackpropL1Reg
[       OK ] LayerUnitTests/1.DenseLayerBackpropL1Reg (0 ms)
[ RUN      ] LayerUnitTests/1.DenseLayerBackpropL2Reg
[       OK ] LayerUnitTests/1.DenseLayerBackpropL2Reg (0 ms)
[ RUN      ] LayerUnitTests/1.DropoutLayerComputeTrain
[       OK ] LayerUnitTests/1.DropoutLayerComputeTrain (0 ms)
[ RUN      ] LayerUnitTests/1.DropoutLayerComputeEval
[       OK ] LayerUnitTests/1.DropoutLayerComputeEval (0 ms)
[ RUN      ] LayerUnitTests/1.DropoutLayerBackprop
[       OK ] LayerUnitTests/1.DropoutLayerBackprop (0 ms)
[ RUN      ] LayerUnitTests/1.InputLayerCompute
[       OK ] LayerUnitTests/1.InputLayerCompute (0 ms)
[ RUN      ] LayerUnitTests/1.InputLayerBackprop
[       OK ] LayerUnitTests/1.InputLayerBackprop (0 ms)
[----------] 10 tests from LayerUnitTests/1 (19 ms total)

[----------] 7 tests from UtilsIntegrationTests/0, where TypeParam = float
[ RUN      ] UtilsIntegrationTests/0.MulMulPow
[       OK ] UtilsIntegrationTests/0.MulMulPow (0 ms)
[ RUN      ] UtilsIntegrationTests/0.MulDivMean
[       OK ] UtilsIntegrationTests/0.MulDivMean (0 ms)
```

```
[ RUN      ] UtilsIntegrationTests/0.MulAddRoot
[       OK ] UtilsIntegrationTests/0.MulAddRoot (0 ms)
[ RUN      ] UtilsIntegrationTests/0.MulSubExp
[       OK ] UtilsIntegrationTests/0.MulSubExp (0 ms)
[ RUN      ] UtilsIntegrationTests/0.DivAddLog
[       OK ] UtilsIntegrationTests/0.DivAddLog (0 ms)
[ RUN      ] UtilsIntegrationTests/0.DivSubAbs
[       OK ] UtilsIntegrationTests/0.DivSubAbs (0 ms)
[ RUN      ] UtilsIntegrationTests/0.AddSubTranspose
[       OK ] UtilsIntegrationTests/0.AddSubTranspose (0 ms)
[----------] 7 tests from UtilsIntegrationTests/0 (0 ms total)

[----------] 7 tests from UtilsIntegrationTests/1, where TypeParam = double
[ RUN      ] UtilsIntegrationTests/1.MulMulPow
[       OK ] UtilsIntegrationTests/1.MulMulPow (0 ms)
[ RUN      ] UtilsIntegrationTests/1.MulDivMean
[       OK ] UtilsIntegrationTests/1.MulDivMean (0 ms)
[ RUN      ] UtilsIntegrationTests/1.MulAddRoot
[       OK ] UtilsIntegrationTests/1.MulAddRoot (0 ms)
[ RUN      ] UtilsIntegrationTests/1.MulSubExp
[       OK ] UtilsIntegrationTests/1.MulSubExp (0 ms)
[ RUN      ] UtilsIntegrationTests/1.DivAddLog
[       OK ] UtilsIntegrationTests/1.DivAddLog (0 ms)
[ RUN      ] UtilsIntegrationTests/1.DivSubAbs
[       OK ] UtilsIntegrationTests/1.DivSubAbs (0 ms)
[ RUN      ] UtilsIntegrationTests/1.AddSubTranspose
[       OK ] UtilsIntegrationTests/1.AddSubTranspose (0 ms)
[----------] 7 tests from UtilsIntegrationTests/1 (0 ms total)

[----------] 4 tests from LayerIntegrationTests/0, where TypeParam = float
[ RUN      ] LayerIntegrationTests/0.DenseReluDenseCCE
[       OK ] LayerIntegrationTests/0.DenseReluDenseCCE (0 ms)
[ RUN      ] LayerIntegrationTests/0.DenseSoftmaxDenseBCE
[       OK ] LayerIntegrationTests/0.DenseSoftmaxDenseBCE (0 ms)
[ RUN      ] LayerIntegrationTests/0.DenseSigmoidDenseMSE
[       OK ] LayerIntegrationTests/0.DenseSigmoidDenseMSE (0 ms)
[ RUN      ] LayerIntegrationTests/0.DenseLinearDenseMAE
[       OK ] LayerIntegrationTests/0.DenseLinearDenseMAE (0 ms)
[----------] 4 tests from LayerIntegrationTests/0 (0 ms total)

[----------] 4 tests from LayerIntegrationTests/1, where TypeParam = double
[ RUN      ] LayerIntegrationTests/1.DenseReluDenseCCE
[       OK ] LayerIntegrationTests/1.DenseReluDenseCCE (0 ms)
[ RUN      ] LayerIntegrationTests/1.DenseSoftmaxDenseBCE
[       OK ] LayerIntegrationTests/1.DenseSoftmaxDenseBCE (0 ms)
```

```
[ RUN      ] LayerIntegrationTests/1.DenseSigmoidDenseMSE
[       OK ] LayerIntegrationTests/1.DenseSigmoidDenseMSE (0 ms)
[ RUN      ] LayerIntegrationTests/1.DenseLinearDenseMAE
[       OK ] LayerIntegrationTests/1.DenseLinearDenseMAE (0 ms)
[----------] 4 tests from LayerIntegrationTests/1 (1 ms total)

[----------] Global test environment tear-down
[==========] 83 tests from 9 test suites ran. (51 ms total)
[  PASSED  ] 83 tests.


[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from SystemTests
[ RUN      ] SystemTests.FashionMnist
Loading train data ...
Loading test data ...
Pre-processing data ...
Constructing model ...
Epoch: 0
step: 0, accuracy: 0.109, loss: 2.302 (data loss: 2.302, reg loss: 0.000), lr: 0.001, time: 16 ms
step: 100, accuracy: 0.727, loss: 0.714 (data loss: 0.714, reg loss: 0.000), lr: 0.001, time: 1445 ms
step: 200, accuracy: 0.812, loss: 0.567 (data loss: 0.567, reg loss: 0.000), lr: 0.001, time: 1462 ms
step: 300, accuracy: 0.812, loss: 0.517 (data loss: 0.517, reg loss: 0.000), lr: 0.001, time: 1454 ms
step: 400, accuracy: 0.828, loss: 0.489 (data loss: 0.489, reg loss: 0.000), lr: 0.001, time: 1447 ms
step: 468, accuracy: 0.802, loss: 1.833 (data loss: 1.833, reg loss: 0.000), lr: 0.001, time: 1000 ms
training, acc: 0.763, loss: 0.651
validation, acc: 0.818, loss: 0.587
Epoch: 1
step: 0, accuracy: 0.805, loss: 0.450 (data loss: 0.450, reg loss: 0.000), lr: 0.001, time: 362 ms
step: 100, accuracy: 0.852, loss: 0.447 (data loss: 0.447, reg loss: 0.000), lr: 0.001, time: 1451 ms
step: 200, accuracy: 0.820, loss: 0.431 (data loss: 0.431, reg loss: 0.000), lr: 0.001, time: 1472 ms
step: 300, accuracy: 0.828, loss: 0.475 (data loss: 0.475, reg loss: 0.000), lr: 0.001, time: 1531 ms
step: 400, accuracy: 0.859, loss: 0.404 (data loss: 0.404, reg loss: 0.000), lr: 0.001, time: 1754 ms
step: 468, accuracy: 0.865, loss: 1.728 (data loss: 1.728, reg loss: 0.000), lr: 0.001, time: 1034 ms
training, acc: 0.844, loss: 0.434
validation, acc: 0.845, loss: 0.524
Epoch: 2
step: 0, accuracy: 0.867, loss: 0.352 (data loss: 0.352, reg loss: 0.000), lr: 0.001, time: 376 ms
step: 100, accuracy: 0.875, loss: 0.385 (data loss: 0.385, reg loss: 0.000), lr: 0.000, time: 1536 ms
step: 200, accuracy: 0.844, loss: 0.382 (data loss: 0.382, reg loss: 0.000), lr: 0.000, time: 1873 ms
step: 300, accuracy: 0.852, loss: 0.418 (data loss: 0.418, reg loss: 0.000), lr: 0.000, time: 1915 ms
step: 400, accuracy: 0.898, loss: 0.345 (data loss: 0.345, reg loss: 0.000), lr: 0.000, time: 1696 ms
step: 468, accuracy: 0.885, loss: 1.746 (data loss: 1.746, reg loss: 0.000), lr: 0.000, time: 1141 ms
training, acc: 0.860, loss: 0.389
```

validation, acc: 0.853, loss: 0.499

Epoch: 3

step: 0, accuracy: 0.867, loss: 0.329 (data loss: 0.329, reg loss: 0.000), lr: 0.000, time: 440 ms

step: 100, accuracy: 0.898, loss: 0.345 (data loss: 0.345, reg loss: 0.000), lr: 0.000, time: 1863 ms

step: 200, accuracy: 0.852, loss: 0.352 (data loss: 0.352, reg loss: 0.000), lr: 0.000, time: 1772 ms

step: 300, accuracy: 0.859, loss: 0.382 (data loss: 0.382, reg loss: 0.000), lr: 0.000, time: 1895 ms

step: 400, accuracy: 0.875, loss: 0.321 (data loss: 0.321, reg loss: 0.000), lr: 0.000, time: 1659 ms

step: 468, accuracy: 0.917, loss: 1.715 (data loss: 1.715, reg loss: 0.000), lr: 0.000, time: 1015 ms

training, acc: 0.870, loss: 0.362

validation, acc: 0.858, loss: 0.481

Epoch: 4

step: 0, accuracy: 0.883, loss: 0.320 (data loss: 0.320, reg loss: 0.000), lr: 0.000, time: 522 ms

step: 100, accuracy: 0.898, loss: 0.324 (data loss: 0.324, reg loss: 0.000), lr: 0.000, time: 1934 ms

step: 200, accuracy: 0.859, loss: 0.323 (data loss: 0.323, reg loss: 0.000), lr: 0.000, time: 2102 ms

step: 300, accuracy: 0.883, loss: 0.362 (data loss: 0.362, reg loss: 0.000), lr: 0.000, time: 1573 ms

step: 400, accuracy: 0.875, loss: 0.311 (data loss: 0.311, reg loss: 0.000), lr: 0.000, time: 1698 ms

step: 468, accuracy: 0.917, loss: 1.713 (data loss: 1.713, reg loss: 0.000), lr: 0.000, time: 1065 ms

training, acc: 0.876, loss: 0.342

validation, acc: 0.864, loss: 0.470

Epoch: 5

step: 0, accuracy: 0.883, loss: 0.315 (data loss: 0.315, reg loss: 0.000), lr: 0.000, time: 408 ms

step: 100, accuracy: 0.891, loss: 0.312 (data loss: 0.312, reg loss: 0.000), lr: 0.000, time: 1599 ms

step: 200, accuracy: 0.867, loss: 0.302 (data loss: 0.302, reg loss: 0.000), lr: 0.000, time: 1654 ms

step: 300, accuracy: 0.906, loss: 0.346 (data loss: 0.346, reg loss: 0.000), lr: 0.000, time: 1472 ms

step: 400, accuracy: 0.875, loss: 0.306 (data loss: 0.306, reg loss: 0.000), lr: 0.000, time: 1467 ms

step: 468, accuracy: 0.917, loss: 1.698 (data loss: 1.698, reg loss: 0.000), lr: 0.000, time: 1000 ms

training, acc: 0.881, loss: 0.328

validation, acc: 0.869, loss: 0.463

Epoch: 6

step: 0, accuracy: 0.898, loss: 0.315 (data loss: 0.315, reg loss: 0.000), lr: 0.000, time: 360 ms

step: 100, accuracy: 0.906, loss: 0.301 (data loss: 0.301, reg loss: 0.000), lr: 0.000, time: 1483 ms

step: 200, accuracy: 0.883, loss: 0.289 (data loss: 0.289, reg loss: 0.000), lr: 0.000, time: 1475 ms

step: 300, accuracy: 0.906, loss: 0.336 (data loss: 0.336, reg loss: 0.000), lr: 0.000, time: 1462 ms

step: 400, accuracy: 0.875, loss: 0.303 (data loss: 0.303, reg loss: 0.000), lr: 0.000, time: 1469 ms

step: 468, accuracy: 0.917, loss: 1.739 (data loss: 1.739, reg loss: 0.000), lr: 0.000, time: 983 ms

training, acc: 0.886, loss: 0.316

validation, acc: 0.870, loss: 0.459

Epoch: 7

step: 0, accuracy: 0.891, loss: 0.312 (data loss: 0.312, reg loss: 0.000), lr: 0.000, time: 361 ms

step: 100, accuracy: 0.906, loss: 0.292 (data loss: 0.292, reg loss: 0.000), lr: 0.000, time: 1467 ms

step: 200, accuracy: 0.875, loss: 0.278 (data loss: 0.278, reg loss: 0.000), lr: 0.000, time: 1495 ms

step: 300, accuracy: 0.914, loss: 0.323 (data loss: 0.323, reg loss: 0.000), lr: 0.000, time: 1496 ms

step: 400, accuracy: 0.875, loss: 0.301 (data loss: 0.301, reg loss: 0.000), lr: 0.000, time: 1454 ms

step: 468, accuracy: 0.927, loss: 1.713 (data loss: 1.713, reg loss: 0.000), lr: 0.000, time: 1001 ms

training, acc: 0.889, loss: 0.307

validation, acc: 0.871, loss: 0.451

Epoch: 8

step: 0, accuracy: 0.898, loss: 0.309 (data loss: 0.309, reg loss: 0.000), lr: 0.000, time: 366 ms

step: 100, accuracy: 0.914, loss: 0.284 (data loss: 0.284, reg loss: 0.000), lr: 0.000, time: 1446 ms

step: 200, accuracy: 0.883, loss: 0.271 (data loss: 0.271, reg loss: 0.000), lr: 0.000, time: 1407 ms

step: 300, accuracy: 0.914, loss: 0.314 (data loss: 0.314, reg loss: 0.000), lr: 0.000, time: 1406 ms

step: 400, accuracy: 0.875, loss: 0.298 (data loss: 0.298, reg loss: 0.000), lr: 0.000, time: 1405 ms

step: 468, accuracy: 0.927, loss: 1.713 (data loss: 1.713, reg loss: 0.000), lr: 0.000, time: 958 ms

training, acc: 0.892, loss: 0.299

validation, acc: 0.873, loss: 0.447

Epoch: 9

step: 0, accuracy: 0.891, loss: 0.308 (data loss: 0.308, reg loss: 0.000), lr: 0.000, time: 348 ms

step: 100, accuracy: 0.914, loss: 0.277 (data loss: 0.277, reg loss: 0.000), lr: 0.000, time: 1420 ms

step: 200, accuracy: 0.891, loss: 0.266 (data loss: 0.266, reg loss: 0.000), lr: 0.000, time: 1400 ms

step: 300, accuracy: 0.914, loss: 0.305 (data loss: 0.305, reg loss: 0.000), lr: 0.000, time: 1403 ms

step: 400, accuracy: 0.891, loss: 0.295 (data loss: 0.295, reg loss: 0.000), lr: 0.000, time: 1396 ms

step: 468, accuracy: 0.927, loss: 1.714 (data loss: 1.714, reg loss: 0.000), lr: 0.000, time: 946 ms

training, acc: 0.893, loss: 0.293

validation, acc: 0.875, loss: 0.446

Epoch: 10

step: 0, accuracy: 0.891, loss: 0.309 (data loss: 0.309, reg loss: 0.000), lr: 0.000, time: 341 ms

step: 100, accuracy: 0.922, loss: 0.269 (data loss: 0.269, reg loss: 0.000), lr: 0.000, time: 1395 ms

step: 200, accuracy: 0.891, loss: 0.262 (data loss: 0.262, reg loss: 0.000), lr: 0.000, time: 1398 ms

step: 300, accuracy: 0.922, loss: 0.297 (data loss: 0.297, reg loss: 0.000), lr: 0.000, time: 1420 ms

step: 400, accuracy: 0.898, loss: 0.292 (data loss: 0.292, reg loss: 0.000), lr: 0.000, time: 1391 ms

step: 468, accuracy: 0.927, loss: 1.747 (data loss: 1.747, reg loss: 0.000), lr: 0.000, time: 950 ms

training, acc: 0.895, loss: 0.287

validation, acc: 0.877, loss: 0.445

Epoch: 11

step: 0, accuracy: 0.891, loss: 0.305 (data loss: 0.305, reg loss: 0.000), lr: 0.000, time: 342 ms

step: 100, accuracy: 0.922, loss: 0.263 (data loss: 0.263, reg loss: 0.000), lr: 0.000, time: 1399 ms

step: 200, accuracy: 0.891, loss: 0.258 (data loss: 0.258, reg loss: 0.000), lr: 0.000, time: 1401 ms

step: 300, accuracy: 0.922, loss: 0.292 (data loss: 0.292, reg loss: 0.000), lr: 0.000, time: 1397 ms

step: 400, accuracy: 0.898, loss: 0.288 (data loss: 0.288, reg loss: 0.000), lr: 0.000, time: 1601 ms

step: 468, accuracy: 0.927, loss: 1.731 (data loss: 1.731, reg loss: 0.000), lr: 0.000, time: 1147 ms

training, acc: 0.897, loss: 0.282

validation, acc: 0.878, loss: 0.441

Epoch: 12

step: 0, accuracy: 0.891, loss: 0.303 (data loss: 0.303, reg loss: 0.000), lr: 0.000, time: 369 ms

step: 100, accuracy: 0.930, loss: 0.259 (data loss: 0.259, reg loss: 0.000), lr: 0.000, time: 1667 ms

step: 200, accuracy: 0.891, loss: 0.252 (data loss: 0.252, reg loss: 0.000), lr: 0.000, time: 1508 ms

step: 300, accuracy: 0.938, loss: 0.287 (data loss: 0.287, reg loss: 0.000), lr: 0.000, time: 1539 ms

step: 400, accuracy: 0.898, loss: 0.285 (data loss: 0.285, reg loss: 0.000), lr: 0.000, time: 1471 ms

step: 468, accuracy: 0.927, loss: 1.722 (data loss: 1.722, reg loss: 0.000), lr: 0.000, time: 994 ms

training, acc: 0.899, loss: 0.277

validation, acc: 0.877, loss: 0.439

Epoch: 13

step: 0, accuracy: 0.883, loss: 0.302 (data loss: 0.302, reg loss: 0.000), lr: 0.000, time: 386 ms

step: 100, accuracy: 0.930, loss: 0.254 (data loss: 0.254, reg loss: 0.000), lr: 0.000, time: 1493 ms

step: 200, accuracy: 0.891, loss: 0.249 (data loss: 0.249, reg loss: 0.000), lr: 0.000, time: 1480 ms

step: 300, accuracy: 0.930, loss: 0.282 (data loss: 0.282, reg loss: 0.000), lr: 0.000, time: 1445 ms

step: 400, accuracy: 0.891, loss: 0.280 (data loss: 0.280, reg loss: 0.000), lr: 0.000, time: 1438 ms

step: 468, accuracy: 0.917, loss: 1.718 (data loss: 1.718, reg loss: 0.000), lr: 0.000, time: 1000 ms

training, acc: 0.900, loss: 0.273

validation, acc: 0.878, loss: 0.438

Epoch: 14

step: 0, accuracy: 0.883, loss: 0.301 (data loss: 0.301, reg loss: 0.000), lr: 0.000, time: 365 ms

step: 100, accuracy: 0.930, loss: 0.250 (data loss: 0.250, reg loss: 0.000), lr: 0.000, time: 1667 ms

step: 200, accuracy: 0.898, loss: 0.245 (data loss: 0.245, reg loss: 0.000), lr: 0.000, time: 1425 ms

step: 300, accuracy: 0.930, loss: 0.277 (data loss: 0.277, reg loss: 0.000), lr: 0.000, time: 1411 ms

step: 400, accuracy: 0.891, loss: 0.276 (data loss: 0.276, reg loss: 0.000), lr: 0.000, time: 1428 ms

step: 468, accuracy: 0.917, loss: 1.710 (data loss: 1.710, reg loss: 0.000), lr: 0.000, time: 953 ms

training, acc: 0.902, loss: 0.269

validation, acc: 0.877, loss: 0.436

[OK] SystemTests.FashionMnist (122368 ms)

[----------] 1 test from SystemTests (122368 ms total)

[----------] Global test environment tear-down

[==========] 1 test from 1 test suite ran. (122368 ms total)

[PASSED] 1 test.

Benchmark Output Log

=== matrix_mul_100x100_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.4759ms

std: 0.0469688ms

Fastest time: 0.427166ms

Slowest time: 0.59425ms

==========

=== matrix_mul_500x500_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 37.352ms

std: 1.80798ms

Fastest time: 35.3172ms

Slowest time: 40.928ms

==========

=== matrix_mul_1000x1000_1000x1000 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 373.304ms

std: 1.83616ms

Fastest time: 369.005ms

Slowest time: 375.941ms

==========

=== matrix_div_100x100_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.009ms

std: 0.000247044ms

Fastest time: 0.009166ms

Slowest time: 0.009333ms

==========

=== matrix_div_500x500_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.1335ms

std: 0.00671969ms

Fastest time: 0.129ms

Slowest time: 0.152125ms

==========

=== matrix_div_1000x1000_1000x1000 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.9014ms

std: 0.0560651ms

Fastest time: 0.858125ms

Slowest time: 1.04304ms

==========

=== matrix_transpose_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.01ms

std: 0.000155327ms

Fastest time: 0.010083ms

Slowest time: 0.010333ms

==========

=== matrix_transpose_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.3061ms

std: 0.0147827ms

Fastest time: 0.294208ms

Slowest time: 0.3485ms

==========

=== matrix_transpose_1000x1000 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 1.6027ms

std: 0.157153ms

Fastest time: 1.41904ms

Slowest time: 1.93892ms

==========

=== matrix_mean_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.001ms

std: 0.000929384ms

Fastest time: 0.001875ms

Slowest time: 0.001959ms

==========

=== matrix_mean_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.0331ms

std: 0.000417596ms

Fastest time: 0.033125ms

Slowest time: 0.034375ms

==========

=== matrix_mean_1000x1000 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.1254ms

std: 0.0022612ms

Fastest time: 0.124625ms

Slowest time: 0.131792ms

==========

=== dense_fprop_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.483ms

std: 0.0474794ms

Fastest time: 0.407959ms

Slowest time: 0.549917ms

==========

=== dense_fprop_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 35.7146ms

std: 0.960861ms

Fastest time: 34.7465ms

Slowest time: 37.48ms

==========

=== dense_backprop_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.6015ms

std: 0.0157596ms

Fastest time: 0.579375ms

Slowest time: 0.630458ms

==========

=== dense_backprop_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 72.9953ms

std: 2.15815ms

Fastest time: 70.2343ms

Slowest time: 77.0604ms

==========

=== dropout_fprop_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.1318ms

std: 0.00217622ms

Fastest time: 0.129125ms

Slowest time: 0.13675ms

==========

=== dropout_fprop_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 3.3178ms

std: 0.00812784ms

Fastest time: 3.30554ms

Slowest time: 3.32979ms

==========

=== dropout_backprop_100x100 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.002ms

std: 0.00055898ms

Fastest time: 0.0025ms

Slowest time: 0.002584ms

==========

=== dropout_backprop_500x500 Benchmark ===

Warming up ...

Running benchmark ...

Avg. execution time: 0.076ms

std: 0.000635107ms

Fastest time: 0.075333ms

Slowest time: 0.077458ms

==========