Karlsruher Institut für Technologie

Implementation Document (FSD)

Numerical Linear Algebra meets Machine Learning

Fabian Koffer

Simon Hanselmann

Yannick Funk

Dennis Leon Grötzinger

Anna Katharina Ricker

Supervisors

Hartwig Anzt Markus Götz

February 6, 2019

Contents

1 Introduction

Goal was the delivery of a consistent software stack that allows for employing neural networks for the linear system. The ecosystem should allow to train a neural network on selecting a suitable iterative solver depending on the linear system characteristics.

Since we were dealing with a big Python project for the first time and we've never heard of Ginkgo before, getting into program environment was the first challenge. It turned out that our design was very good and we hardly had to make any changes.

Overall, our project now consists of around 1500 lines of Python code and around 200 lines of C++ code.

2 Changes on the Design

- We didn't implemented a matrix class, because we realized it is not useful to handle every matrix on its own. Instead we decided to handle it in numpy arrays.
- We changed the command interface to an abstract class to reduce redundancy of the code.
- We decided to let out the possibility to select a density, because we couldn't guarantee that the fetched and cut matrices from Suite Sparse have the wanted density.
- We added a help command, so the user is able to get an overview which commands can be executed.
- We added a ssget command so the user is able to update the file, where all the matrices ID's are listed. So when the Suite Sparse collection is extended the list can be updated to include the new matrices.

3 The Requirements

3.1 Following Requirements are accomplished

- A software that supports the described work-flow design including the embedding of external components.

- The software must be usable via a command-line interface (CLI).
- A data exchange format design that allows to store matrices and annotate them with additional meta-data, including labels.
- An extensible design for multiple entities that are able to generate matrices in the proposed exchange format.
- A dataset of at least 500 matrices in the envisioned data format and generated by the above two entities. There smallest share of matrices of a given entity must be no less than 30% of the total number of contained matrices.
- An extensible design that allows to solve the matrices using a configurable set of iterative solver algorithms using a newly developed binding to the Ginkgo linear algebra library.
- A readily implemented and trained neural network of the resNet architecture. It must be able to predict for a given matrix (in arbitrary format), which of the iterative solver algorithms is the most suitable.
- An entity that allows to store and load the trained neural network.
- The software must include entities for training and re-training a neural network from scratch, respectively from a previously stored state.
- The software must be able to show the predicted algorithm and its associated suitability probability on the standard output.
- Realization of a sustainable and quality-assured software development process. This includes a software design document, in-code documentation, unit testing and a continuous integration (CI).

Following optional requirements are accomplished:

- The software must be able to utilize GPU accelerators for the training and prediction capabilities of the neural network.

3.2 Following Requirements were accomplished as far as possible

- All mandatory requirements were as far as possible accomplished despite the cross-platformed compatibility is not fully given. This was not possible, because some used entities (ssget, gingko) where not Windows compatible.

- Compared to the specification sheet there is just the possibility to fetch and cut Suite Sparse matrices yet. As we figured out our design we realized that generating random matrices is not that easy with our knowing. So as already realized in the design document, because in Suite Sparse it is very rare to have same sized matrices from Suite Sparse, it was not necessary to just fetch Suite Sparse matrices in one size. Instead we just implemented a generator that fetches and cuts Suite Sparse matrices.

- We decided to let out the possibility to select a density as explained in the changes of design

- The system just supports four instead of five iterative solver algorithms. One solver was not working properly so for now we decided just to use the four working solver algorithms.

3.3 Following optional Requirements were not accomplished

- A web interface to the software that is able to select a single, a set or all matrices of an uploadable file for prediction by the neural network. The web interface may also be able to visualize the contained matrices, annotated labels as well as prediction results.

- Scalability of the workflow including matrix generation, training, prediction in that multiple processors may be used in parallel.

4 Unittests

4.1 Controller

4.1.1 test_command_parser

This test checks the functionality of the command parser. It has following tests:

- test_valid_input_returns_command

- test_valid_input_with_arguments

- test_valid_input_with_flag

- test_invalid_mode_throws_exception
- test_valid_collector_input
- test_valid_label_mode
- test_fails_when_entering_invalid_module
- test_quit_with_arguments_throws_error
- test_collector_with_missing_optional_args_adds_default
- test_classify_command_with_missing_optional_arg_adds_default

4.1.2 test_controller

This test checks the functionality of the controller. It has following tests:

- test_controller_with_two_iterations
- test_invalid_input_calls_print_error
- test_help_flag_print
- test_ssget_update_command_calls_new_search

4.2 model

4.2.1 test collector

This test checks the functionality of the collector. It has following tests:

- test_collect

4.3 shared

4.3.1 test configurations

This test checks the functionality of the loading of the configurations. It has following tests:

- test_loading_config_values_works
- test_loading_config_has_right_value

4.4 view

4.4.1 test cli output service

This test checks the functionality of the view. It has following tests:

- test_create_observable_to_print_three_values

5 Delays and Problems

5.1 Labeling module

Our main problem was with the Ginkgo. It was barely documented what made working with it very hard and caused a delay in our implementation plan.

First we had to find out, how to use Ginkgo and because there was no good documentation we just had examples of the repository to work with.

The next difficulty was to integrate C++. We didn't managed to find that out by our own, but with ctypes and the help of markus (he wrote us a tutorial) we could integrate C++.

After that we needed to figure out how to transfer the data to C++. With pointer in python, csr matrix format and more examples from Ginkgo repository we also solved this Problem.

The following step was to find out how the server we use works so we can find out how to use Pycharm on the server. This we got to work with a remote interpreter.

The high resolution clock function in C++ solved our next problem: the determination of the time the solvers need.

The last problem was, that the cuda server wasn't working, because there was a bug in Ginkgo on the Server. So Markus has to help us again to fix that.

After that we just had to clean up the methods to let them pass the codeclimate tests. Therefore we tried to outsource functions, especially our function that chooses which solver is to be selected. That we tried this by mapping with pointers. Because here the return type was not possible, so it didn't work, we finally handled the problems with templates, how Markus showed us.

5.2 MVC

The Basic structure for MVC was very easy to implement. Through preliminary work in the implementation phase, it was very easy to build the basic structure However, there were some problems with integration with real modules later.

5.3 Training module

The problem with the labeling module caused a delay in the implementation of the training and classify module. testing the training / classify module, was difficult, because the data from labeling modules were needed.

5.4 Collector

As we started implementing the collector our first barrier was ssget. Finding out how to download matrices was not the problem. But every matrix itself was saved on different places in the different .mat files, so the only way to find the matrices was with implementing a method that finds the file type every matrix is saved in.

6 Lessons learned

First of all we thought the machine learning part of this software will be the hardest part. But it turned out to be on of the easiest. Instead of that Ginkgo was the main problem. For other projects it probably will be easier to use of a better documented library. On the other hand did we need this specific GPU accelerated solvers, so there probably would have been no better or similar solution.

It was also a lesson that it is not worth to spend much time in data preprocessing, because keras offers many possibilities.

Merging the LaTex documents was a problem, because we didn't had a good LaTex-Editer that supports versioning tools like GIT. Also we sometimes didn't start a new line for each sentence which resulted in bad readability on the pull requests on GitHub. In future it would be better to use for example ShareLaTeX that allows you to edit the same project simultaneously.

7 Statistics

- lines of Python code: 1500
- lines of C++ code: 200
- commits: 160
- test coverage: 80%

7.1 Work Splitting

- Collector module: Yannick and Anna
- Labeling module: Fabian and Dennis
- Training module: Yannick
- Classifier: Yannick
- Command parsing module: Simon

- Output service module: Simon
- Help classes: Yannick
- Implementation report: Anna

8 Development model

8.1 Communication in the team

For talking about the progress and following steps the whole team met one to two times a week and another time with the tutor. Dennis and Fabian worked on the Labeler module together while Yannick and Anna implemented the Collector. Simon did the command parsing module and the output service module. As soon as the labeler module worked, Yannick implemented the training module and the classifier.

8.2 Git

For the previous steps, we already set up a GitHub project where we all committed our work on. Merging on the master branch was restricted from the beginning on. If you think your changes are ready to be pushed to the master, you need to push your feature branch that is up-to-date with the master branch. On GitHub you can then create a pull request for this feature branch. This will create a notification on Slack. Now one of the colleagues needs to review the pull request together with all the changes that happened to the code. He can either suggest some changes or, if he thinks everything is alright, approve the changes. For the implementation phase we also added two automatic checks that will run on all pull requests. The first one is a CI-Tool which runs all tests. If this test fails, the branch can't be merged into the master. The other one is a code checking tool, which checks complexity and readability of the code. This test is optional, but we still tried to keep this test passing.

8.3 Continuous Integration

For continuous integration we decided on Travis-CI. That is because it works very well with GitHub and also offers free workers for the build process. The build consists of the following steps:

1. download and install SSGet
2. install required packages for Python
3. download test reporter for test coverage report
4. run pytest-cov and create a file with the report
5. upload report to code climate

Travis will also send a report to the pull request on GitHub to show if the build and tests were successful. This helped to make sure that only code can be merged to the master, that doesn't brake existing tests.

8.4 Code Checking

Besides the continuous integration, we also integrated CodeClimate. CodeClimate is a code checking tool that looks at the code and uses some metrics to decide how readable and complex the code is. This tool is triggered when someone creates a pull request. It then will look at all the code that differs from the master branch and creates a report about all the problematic code passages that where added. We also added the PEP8 plugin to verify that our code is also PEP8 conform. The CodeClimate check was optional for a pull request, but we still tried to keep this test passing as well. We kept it optional, because for some code passages, like the C++ files, we didn't quite have the time to clean up this code. It still helped a lot to keep our code lean and readable. Thanks to the Travis test report, CodeClimate also shows the current test coverage on each pull request together with its difference to the master branch. When visiting the code climate website, you can also get a little summary of the code base of the master branch.

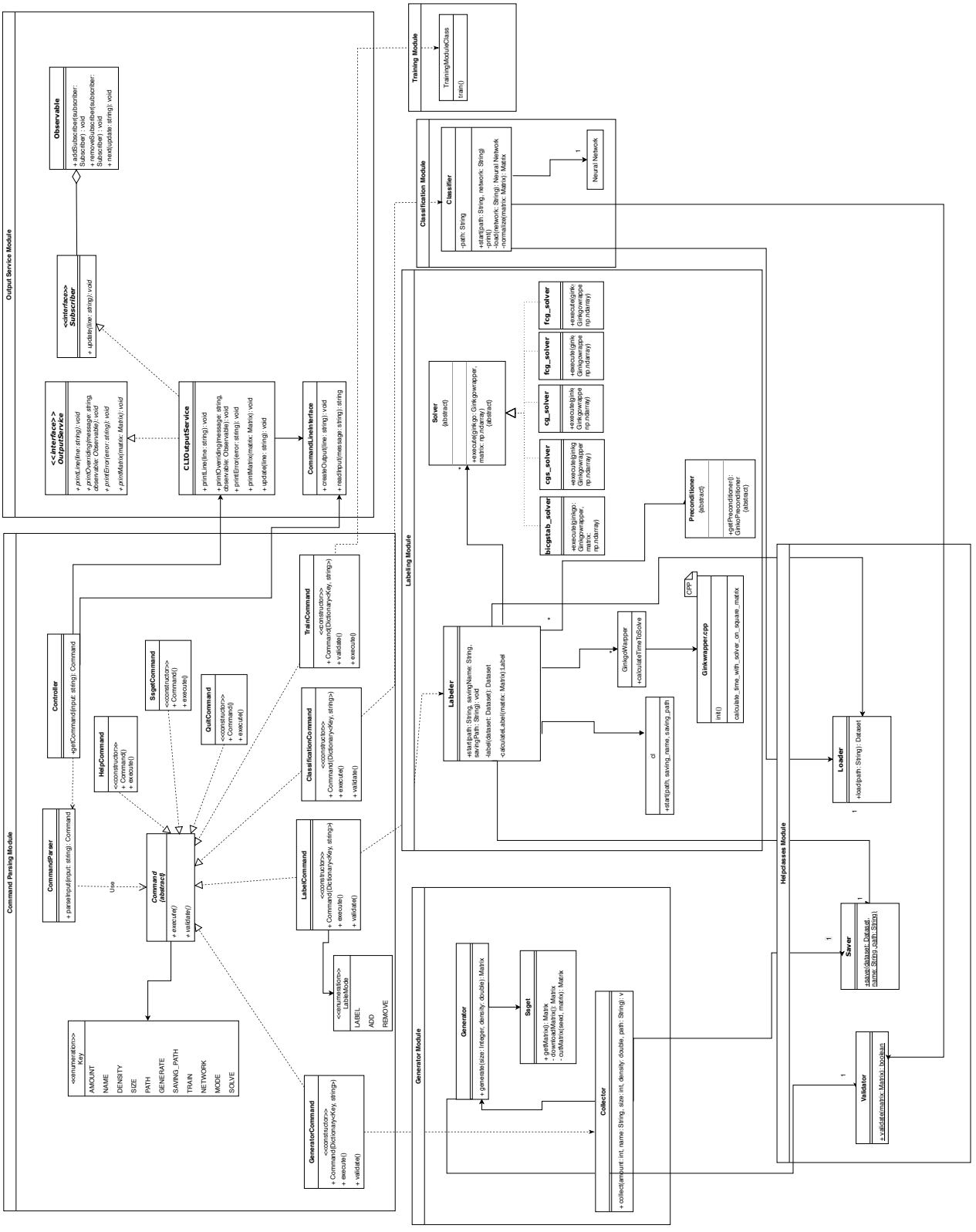

Figure 1: Big class diagram after design changes

9 Glossary

Glossary

algorithm In mathematics and computer science, an algorithm is an unambiguous specification of how to solve a class of problems. Algorithms can perform calculation, data processing and automated reasoning tasks.

command-line interface A command-line interface is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines). A program which handles the interface is called a command language interpreter.

Ginkgo Ginkgo is a high-performance linear algebra library for manycore systems, with a focus on sparse solution of linear systems.

iterative solver In computational mathematics, an iterative solver does a mathematical procedure that uses an initial guess to generate a sequence of improving approximate solutions for a class of problems, in which the n-th approximation is derived from the previous ones.

neural network The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

resNet A deep residual network (deep ResNet) is a type of specialized neural network that helps to handle more sophisticated deep learning tasks and models.

ssget Ssget is a command line tool for downloading matrices from the Suite Sparse Matrix Collection.