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

January 30, 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 lines of Python code and 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 by 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.

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).

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, because we couldn't guarantee that the fetched and cut matrices from Suite Sparse have the wanted density.

3.3 Following Requirements were not accomplished

4 Unittests

4.1 Controller

4.1.1 test command parser

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

- dicts equal
- 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

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

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

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. The same time editing of latex documents was another issue. In future it would be better to use for example ShareLaTeX and start a new line for each sentence to minimize git problems.

7 Statistics

- lines of Python code
- lines of C++ code
- commits

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

- 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

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.