Karlsruher Institut für Technologie

Functional Specification Document (FSD)

Numerical Linear Algebra meets Machine Learning

Fabian Koffer

Simon Hanselmann

Yannick Funk

Dennis Leon Grötzinger

Anna Katharina Ricker

Supervisors

Hartwig Anzt Makus Götz

16. November 2018

1 Success Criteria

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

1.1 Mandatory Requirements

- A software that supports the described work-flow design including the embedding of external components.
- The software must be cross-platform compatible and support at least a Linux and the Windows operating system.
- 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.
- There need to be two actual realizations of these entities, which:
  - allow to generate artificial noise with uniform and gaussian noise as well as
  - can fetch test matrices from the Suite Sparse matrix collection.
- 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).

1.2 Optional Requirements

- Scalability of the work-flow including matrix generation, training, prediction in that multiple processors may be used in parallel.
- The software must be able to utilize GPU accelerators for the training and prediction capabilities of the neural network.
- The system must support at least five iterative solver algorithms.
- 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.

1.3 demarcation criteria

- no matricies other that sparse, squared matricies will be supported

2 Product use

2.1 Scope of application

The software will be used for scientific work in the field of maths and computer science.

2.2 Target groups

Mathematicians and computer scientists who are working with sparse linear systems.

2.3 Operating conditions

- Use in the field of scientific work
- Office environment

3 Product enviroment

3.1 Software

- The product will run on Windows 10 and Linux distributions
- The labeling of the matrices and training of the neural network will be done with Linux

3.2 Hardware

- The product will run on a workstation computer
- The labeling of the matrices and training of the neural network will be done on a server with multiple GPUs

3.3 Orgware

A Documentation for the user will be generated.

4 Functional requirements

4.1 Matrix Generation

- /F10/ Generation of sparse matrices by a given sparsity level and size
- /F15/ Generation of a given amount of matrices
- /F30/ Putting noise on the matrices
- /F40/ Saving the generated matrices in a given directory

4.2 Matrix Labeling

- /F50/ Choice of data origin(local or ssget tool)
- /F60/ Integration of the ssget tool
- /F70/ Determination of the best solving algorithm by time(fix algorithms)
- /F71/ optional: Determination of the best solving algorithm by time with custom algorithms
- /F75/ Labeling the matrix with the determined best algorithm
- /F90/ Creating a grayscale sparsity pattern image of the labeled matrix
- /F100/ Saving the labeled matrix with its sparsity pattern image in a given directory

4.3 DNN Training and Testing

- /F110/ Input of matrix files from a given directory
- /F120/ Randomization of the matrix files order
- /F125/ Separation of the matrix files into a training and testing dataset
- /F130/ Existence of a neural network to train

- /F140/ Training of the neural network by a given training dataset(/F125/)
- /F141/ Testing of the neural network by a given testing dataset(/F125/)
- /F150/ Printing the accuracy(/loss) during the training and testing process of the neural network
- /F151/ optional: creating of accuracy histograms
- /F160/ Saving the neural network in its current state on a given directory

4.4 Matrix Classification

- /F170/ Input of a matrix to classify
- /F175/ Creating a grayscale sparsity pattern image of the input matrix
- /F180/ Loading a trained neural network from a given directory
- /F200/ Classification of the given pattern image by the neural network
- /F201/ Printing the classification output

5 Product data

- 500-1000 matrices for training, validation and test

6 Nonfunctional requirements

- /NF10/ All matrices are squared
- /NF20/ All fixed algorithms are used within the labeling processes
- /NF30/ The grayscale sparsity pattern all have the same size before training
- /NF40/ The NN ouputs a distinct prediction

- /NF50/ The NN is saved after every iteration

7 Global test cases

7.1 the following function sequences must be checked

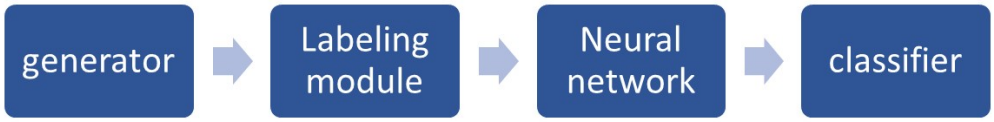
7.2 the following data consistencies must be checked

8 System models

8.1 Scenarios

8.1.1 Overview

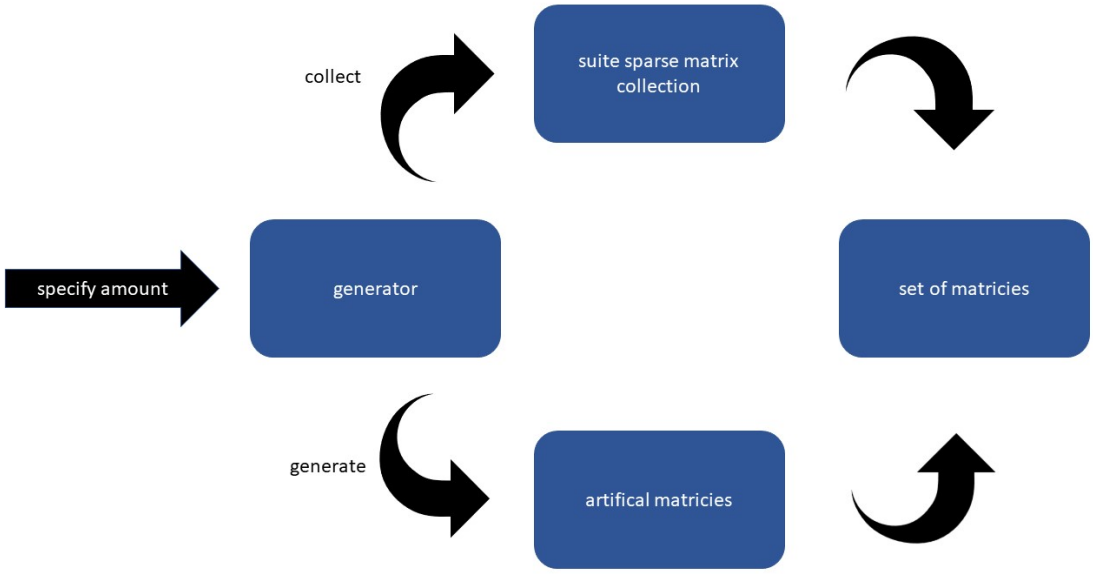
The product consists of four individual modules. The main module is the classifier . Here the user may input a sparse matrix and receive the fastest preconditioner/iterative solver combination for solving the matrix. The user may furthermore interact with the other three modules; the generator, the labeling module and the neural network. Typical interaction with those modules will be described below.



8.1.2 Use of the generator

Optional: The user wants a set of matrices for his own purpose. He likes our idea of combining the creation of matrices with fetching matrices from the suite sparse matrix collection. He therefore opens the module generator. With the command generate <amount> the specified amount of matrices will be created/collected. With this command the ratio will be determined by the program. With the command create <amount> the specified amount of matrices will be created. With the command collect <amount>

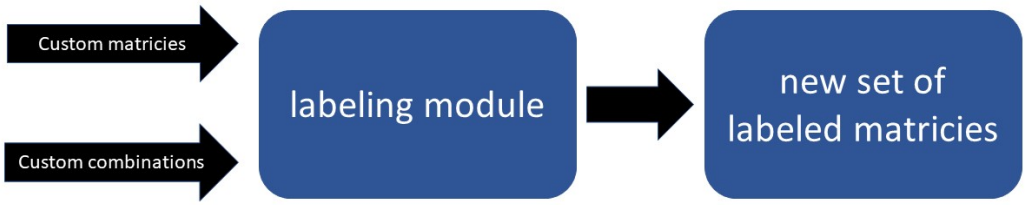
the specified amount of matrices will be collected. After each command the user will be asked to specify a directory to save the matrices to.



8.1.3 Use of the labeling module

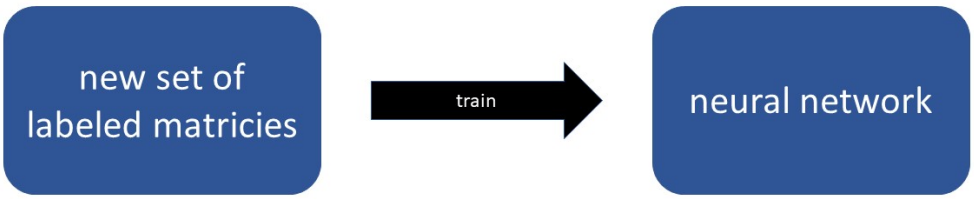
Optional: The user has a very specific problem which generates only a certain kind of sparse matrices. That is why he wants to adapt the neural network for his specific task. He furthermore only wants to use the default preconditioner/iterative solver combinations. He first of all safes all of his matrices in one directory of his choice. Then he opens the labeling module. The command label <filepath> will label all the matrices in the directory. The user may proceed with using the neural network.

Optional: The user is fine with the default matrices, but he wants to use other preconditioner/iterative solver combinations which are included in the ginkgo libary. He opens the labeling module. With the command list combinations he will see all the combinations that are currently used. With the command add <preconditioner/iterative solver> a the new combinations of his choice will be added. With the command delete <preconditioner/iterative solver> the specified combination will be deleted. After the user made his choice he may proceed with using the neural network.



8.1.4 Use of the neural network

Optional: The user has changed the set of matrices and/or the preconditioner/iterative solver combinations. He now wants to build the classifier. He opens the neural network module. With the command train the neural network will be trained. The user may then proceed with using the classifier.



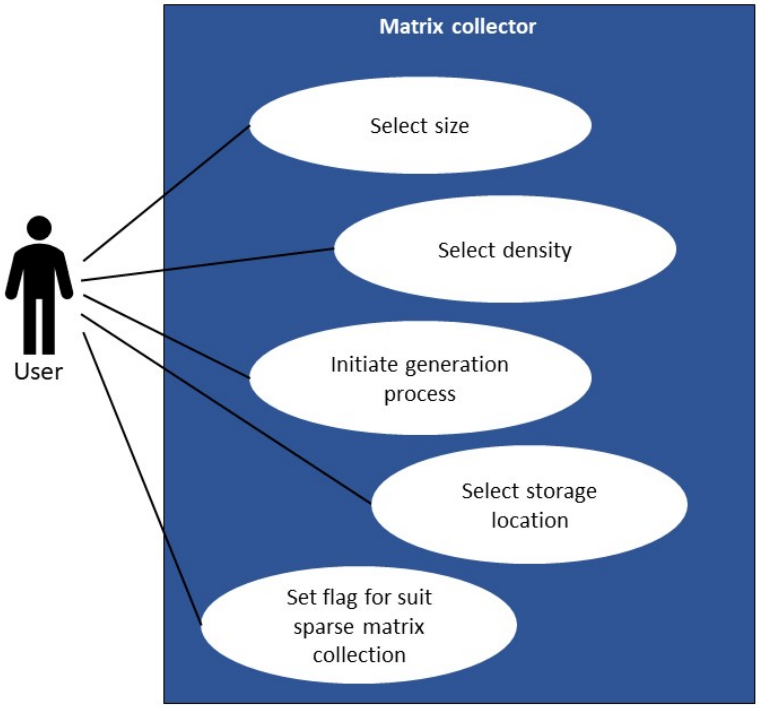
8.1.5 Use of the classifier

The user wants to find the fastest preconditioner/iterative solver combination for a sparse linear system. If he did not change anything in the previous modules the default settings will be used. He will first save the sparse matrix in any desired filepath. Afterwards the user starts our program. The command open <filepath> loads the matrix into the program. That is when the neural network will classify the matrix and determine the fastest iterative solver/preconditioner combination for solving the matrix. This combination will be printed to the command line. (**Optional:** After determining the fastest combination the program will solve the matrix with this combination. The solved matrix will be safed in a directory the user specifies.)

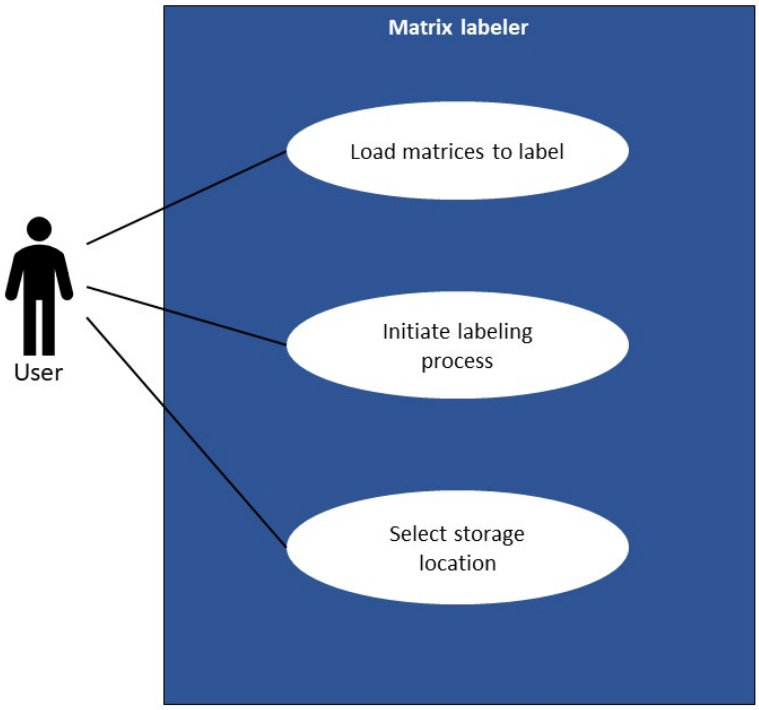

Classifier
“Use iterative solver/preconditioner combination XY“

8.2 Use cases

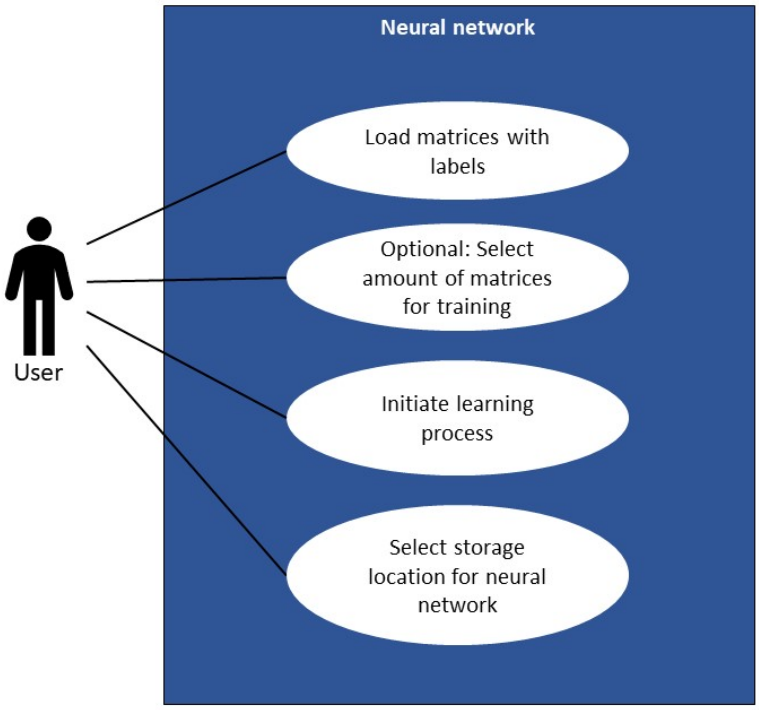
8.2.1 Matrix collector



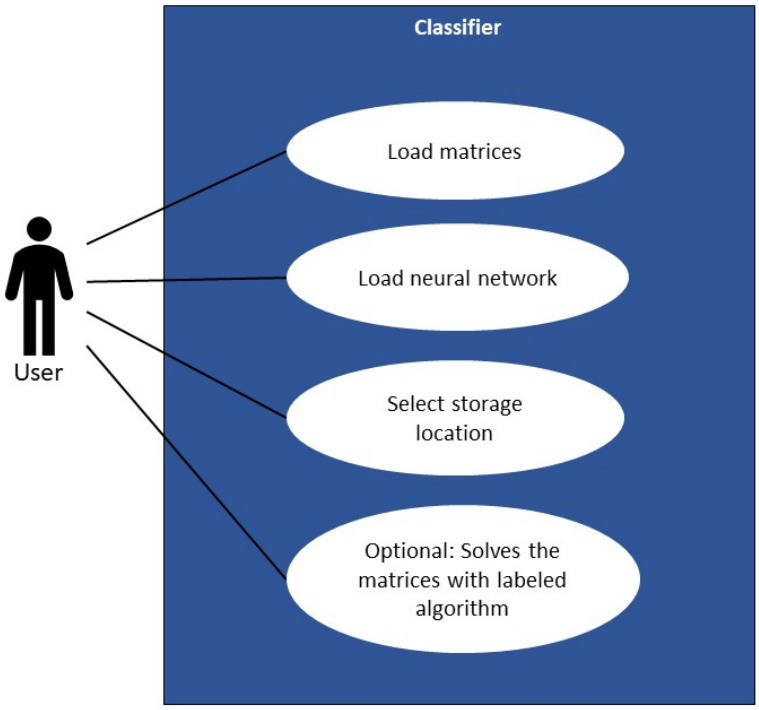
8.2.2 Matrix labeler



8.2.3 Neural network



8.2.4 Classifier



8.3 Object models

8.4 dynamic models

8.5 Command line options

- /B10/**collect -a <amount> -n <name> -s <size> -d <density> -g**:
  The user is able to create a specified amount of matrices that will be saved under a given name

  Arguments

  - **-a <amount>** Absolute amount of matrices the user wants to generate

  - **-n <name>** Name under which the matrices will be saved

- **-s <size>** (optional) Absolute size the generated square matrices should have. Default is 128

- **-d <density>** (optional) Density level of the matrices. A float between 0 and 1 where 1 means no zero values

- **-g**(optional) (Flag) If set it downloads as many matrices of that size as possible from the suite sparse matrix collection

Print

- Progress notifying about the amount of matrices that are created and still need to be created

- A message when process has finished with the path to the created matrices

- Error, in case any required arguments are missing or invalid

- Error, in case the specified name is already taken

- Error, in case **-g** is set and user has no internet connection

- /B20/**label -p <path> -n <name>**:
The user is able to pass matrices that he wants to get labeled

Arguments

- **-p <path>** Absolute path to the matrices in the local storage the user wants to have labeled

- **-n <name>** Name under which the labeled matrices will be saved

Print

- Progress notifying about the amount of matrices that are labeled and still need to be labeled

- A message when process has finished with the path to the labeled matrices

- Error, in case any required arguments are missing or invalid

- Error, in case matrices have wrong format

- Error, in case the specified name is already taken

- Error, in case the remote fetching of the matrices did result in an error

- /B30/**train -p <path> -n <name> -t <train>**:
  The user is able to pass labeled matrices to a neural network, that will learn from this matrices

  Arguments

  - **-p <path>** Absolute path to the labeled matrices on the local storage

  - **-n <name>** Name under which the neural network will be saved after training has finished

  - **-t <train>** (optional) Float between 0 and 1. Amount of matrices used for training where 1 means all. Standard is 0.8

  Print

  - Progress notifying about the loss of the current state based on test data

  - A message when process has finished with the path to the neural network and the final loss

  - Error, in case any required arguments are missing or invalid

  - Error, in case matrices have wrong format or are not labeled

  - Error, in case the specified name is already taken

- /B40/**classify -p <path> -n <network> -s**:
  The user is able to pass a matrix to a the trained neural network, which will find the best solving algorithm.

  Arguments

  - **-p <path>** Path to the matrix the user wants to classify

  - **-n <network>** Path to the trained neural network

  - **-s** (optional) (Flag) If set matrix will also be solved after classification.

  Print

  - The algorithm which will be the fastest on the given matrix

- (optional) The solved matrix
- Error, in case any required arguments are missing or invalid
- Error, in case the matrix has a wrong format
- Error, in case the neural network or matrix path is wrong

9 Glossar

Glossar

classifier The last and main module in the program.It is able to determine the fastest preconditioner/iterative solver combination for a given sparse linear system. It uses the neural network trained by the module neural network..