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

November 30, 2018

Contents

1 Success Criteria

Goal is 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.

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 matrices other than sparse, square matrices will be supported

- It should not be possible to program your own algorithms. Only access to existing algorithms is available.

- Except for the matrices of Suite Sparse, no other collections of matrices 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 environment

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 Collecting

- /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
- /F50/ Choice to either only generate matrices or to generate some and fetch some from the Suite Sparse matrix collection
- /F60/ Integration of the ssget tool

4.2 Matrix Labeling

- /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 grayscale sparsity pattern image in a given directory

4.3 Deep neural network 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/ Loading a neural network from a given directory
- /F161/ 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 grayscale sparsity pattern image by the neural network
- /F201/ Printing the classification output

5 Product data

- artificial generated matrices

- collected matrices by ssget
- 500-1000 labled matrices for training and test
- grayscale sparsity pattern image
- neural network

6 Nonfunctional requirements

- /NF10/ All matrices are square
- /NF20/ All fixed algorithms are used within the labeling processes
- /NF30/ The grayscale sparsity pattern images all have the same size before training
- /NF40/ The neural network ouputs a distinct prediction
- /NF50/ The neural network is saved after every iteration

7 Global test cases

7.1 the following function sequences must be checked

7.1.1 Matrix Generation

- /T10/ When generating a matrix, the returned matrix must have the right size and sparsity level
- /T15/ When generating a matrix, it must be square
- /T20/ When generating an amount of matrices, there must be an exact number of matrices by the given amount
- /T30/ When putting noise on a matrix, the returned matrix must have noise
- /T40/ After saving a matrix in the given directory, there must be the right file, with the right format

7.1.2 Matrix Labeling

- /T50/ When labeling a Matrix, the label must be the algorithm with the shortest time needed
- /T60/ When creating a grayscale sparsity pattern image it must have the same size as the matrix and must actually be grayscale
- /T70/ After saving the labeled matrix with its grayscale sparsity pattern image in a given directory, there must be a file including the right matrix, the corresponding grayscale sparsity pattern images and the right label

7.1.3 Deep neural network Training and Testing

- /T80/ When a matrix is loaded from a given directory, the right matrix must be loaded
- /T90/ When the matrix files are randomized they have to be in a different order after every randomization
- /T100/ After separating the dataset into a training and testing dataset, the training dataset must contain more matrices than the testing dataset
- /T110/ The neural network has the right input dimensions for being trained by the matrices
- /T120/ The neural network is only trained by the training dataset
- /T130/ The neural network is only tested by the testing dataset
- /T140/ The printed loss(/accuracy) is the actual loss of the current training(/testing) iteration
- /T150/ After saving the neural network in its current state, there must be the right file in the right directory

7.1.4 Matrix Classification

- /T160/ After the input of a Matrix, the right file is loaded

- /T170/ After loading a neural network from a given directory, the right neural network has to be in the system

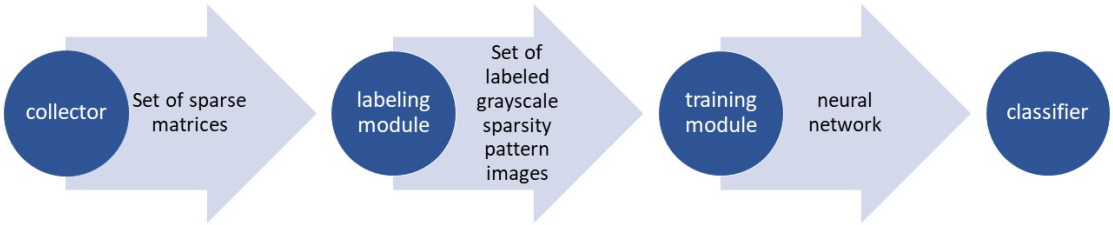
- /T180/ When printing the classification output, it has to be the prediction of the neural network

8 System models

8.1 Scenarios

8.1.1 Overview

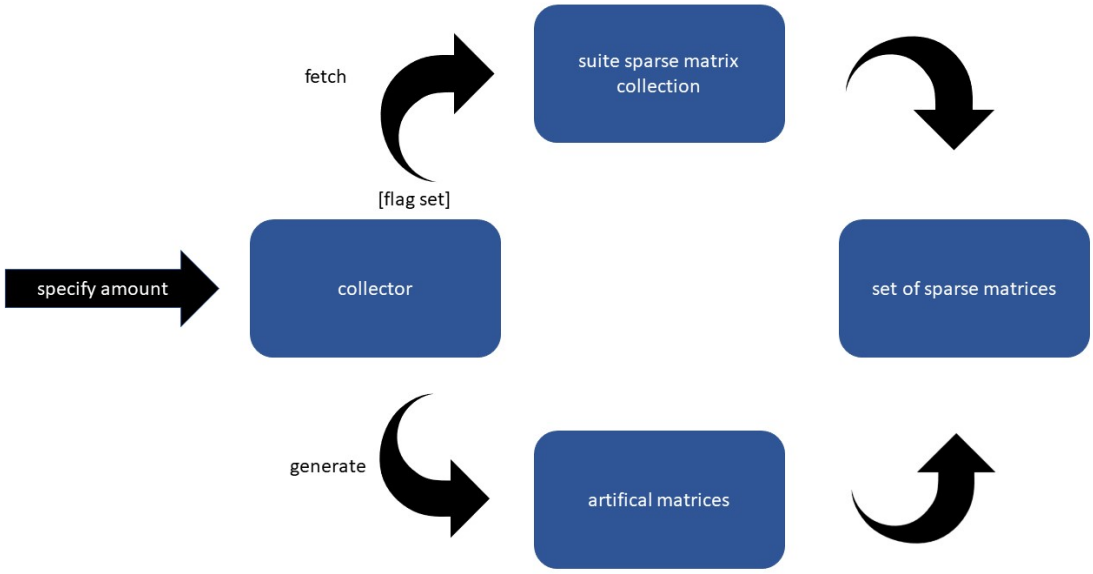
The product consists of four individual modules. The main module is the classifier . Here the user may input a sparse matrix and receive the fastest preconditioner/iterative solver combination for solving the matrix. The user may furthermore interact with the other three modules; the collector, the labeling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



8.1.2 Use of the collector

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 collector. He wants to generate 300 matrices of size 64x64 with a density of 0.2 and wants to use as many matrices as possible from the Suite Sparse matrix collection. He wants to call his set of matrices collectedMatrices and enters the command *collect -a 300 -n collectedMatrices -s 64 -d 0.2 -g*. When the process finished

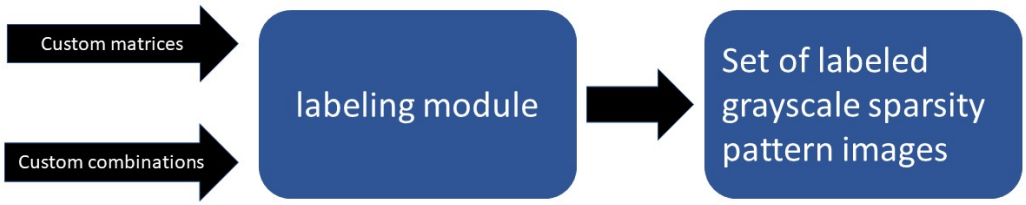
he gets notified. He navigates to the results file of the collector in the file manager and proceeds with using the set of matricies for his own purposes.



8.1.3 Use of the labeling module

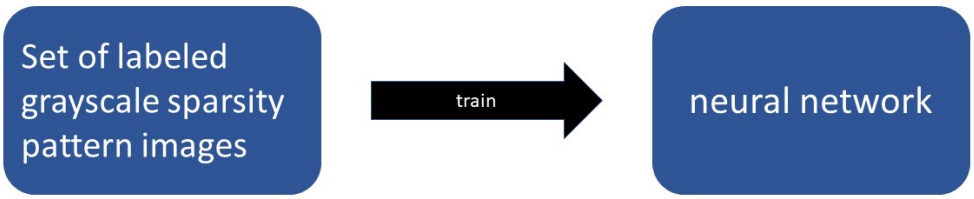
The user has a very specific problem which generates only a certain kind of sparse matrices. That is why he wants to adapt the neural network for his specific task. He furthermore only wants to use the default preconditioner/iterative solver combinations. He first of all safes all of his matrices in one directory x of his choice. Then he opens the labeling module. The command *labeler label -p x -n labeledMatrices* will label all the matrices and generate their grayscale sparsity pattern image in the directory. The set of grayscale sparsity pattern image will be safed with the name labeledMatrices in the results file of the labeling module. 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 *labeler list* he will see all the combinations that are currently used. With the command *labeler add <preconditioner/iterative solver>* the new combinations of his choice will be added. With the command *labeler remove <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 training module

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 training module. With the command *train -n myNeuralNetwork* the neural network will be trained. The neural network will be saved in the results folder with the name myNeuralNetwork. The user may then proceed with using the classifier.



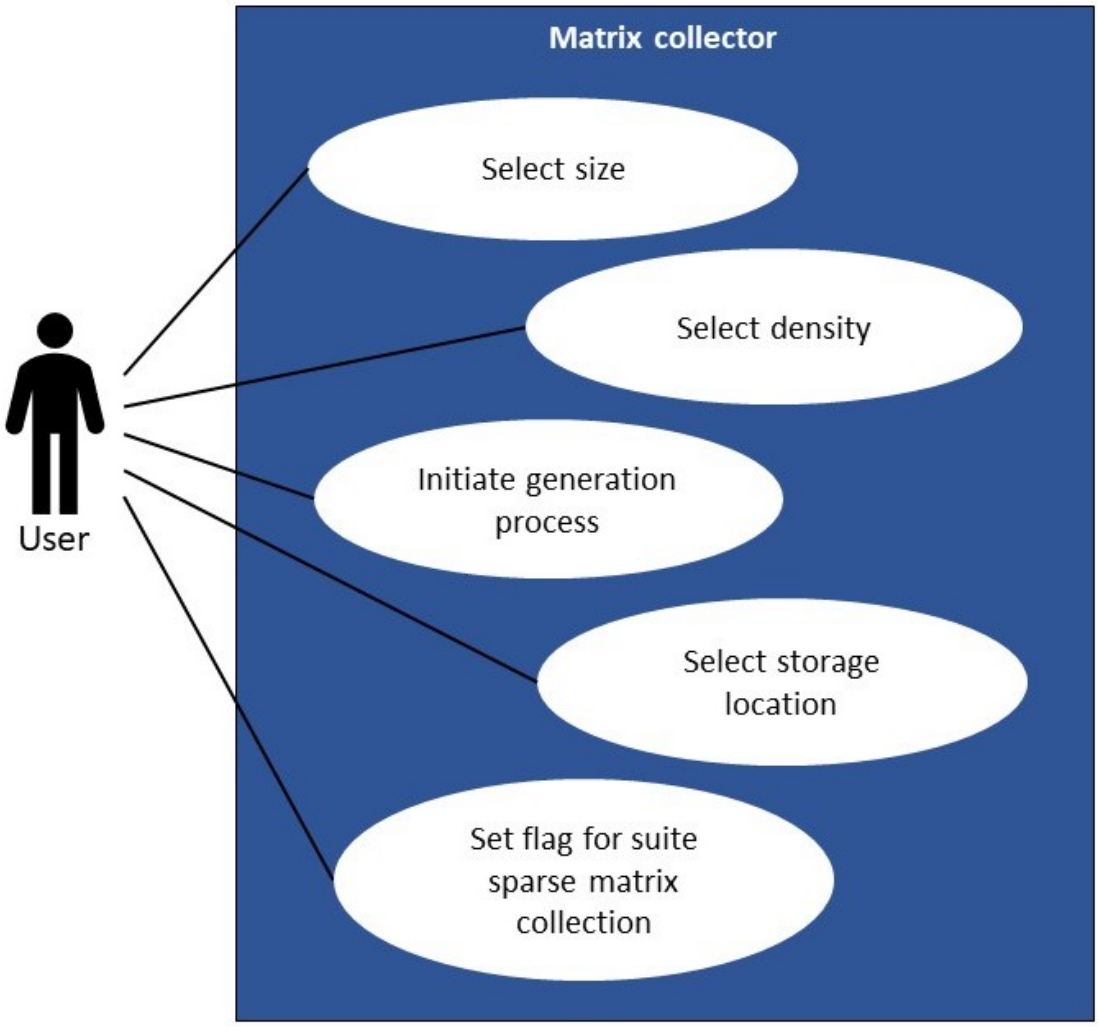
8.1.5 Use of the classifier

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 x. Afterwards the user starts the classifier. With the command *classify -p x - 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.)

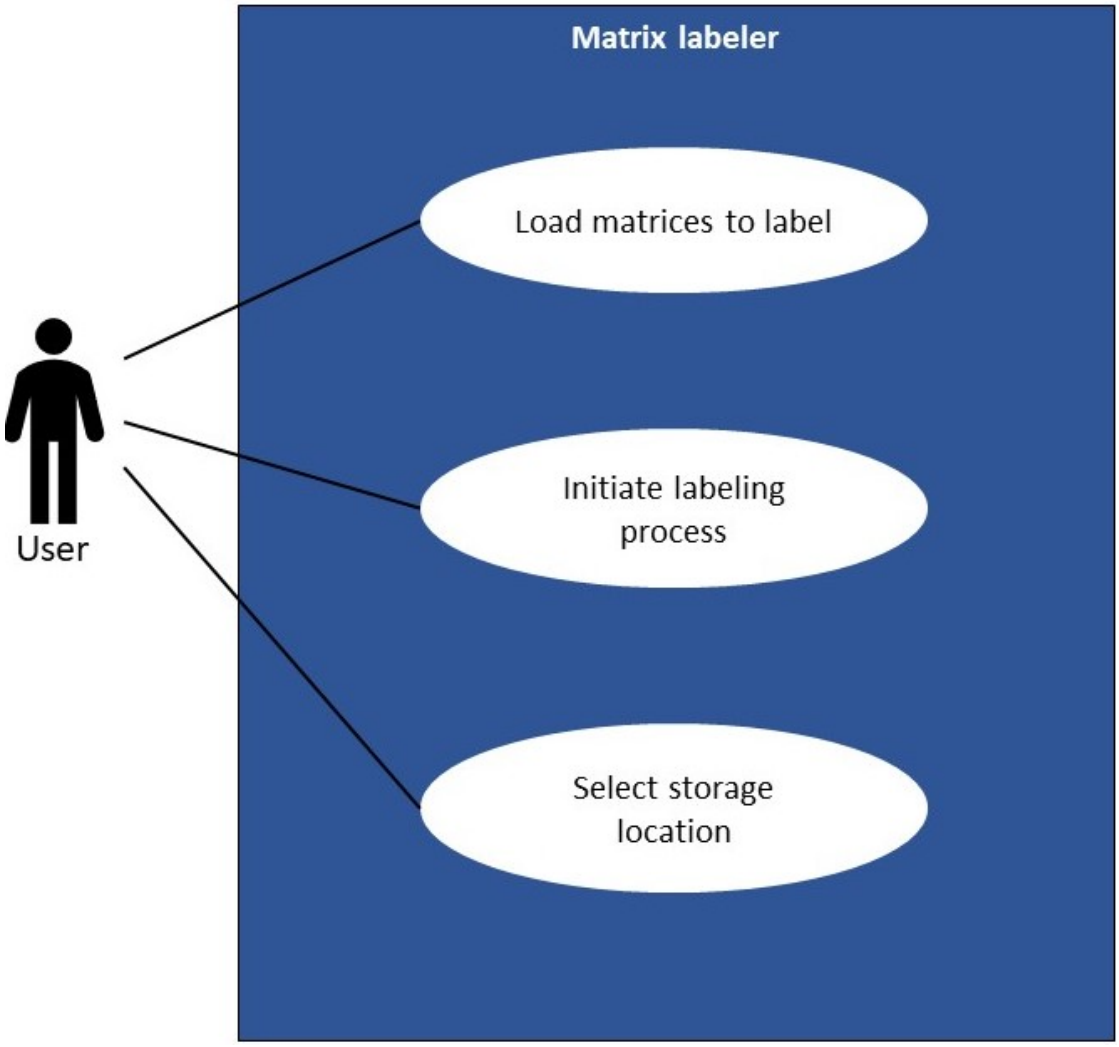

Classifier
“Use preconditioner/iterative solver combination XY“

8.2 Use cases

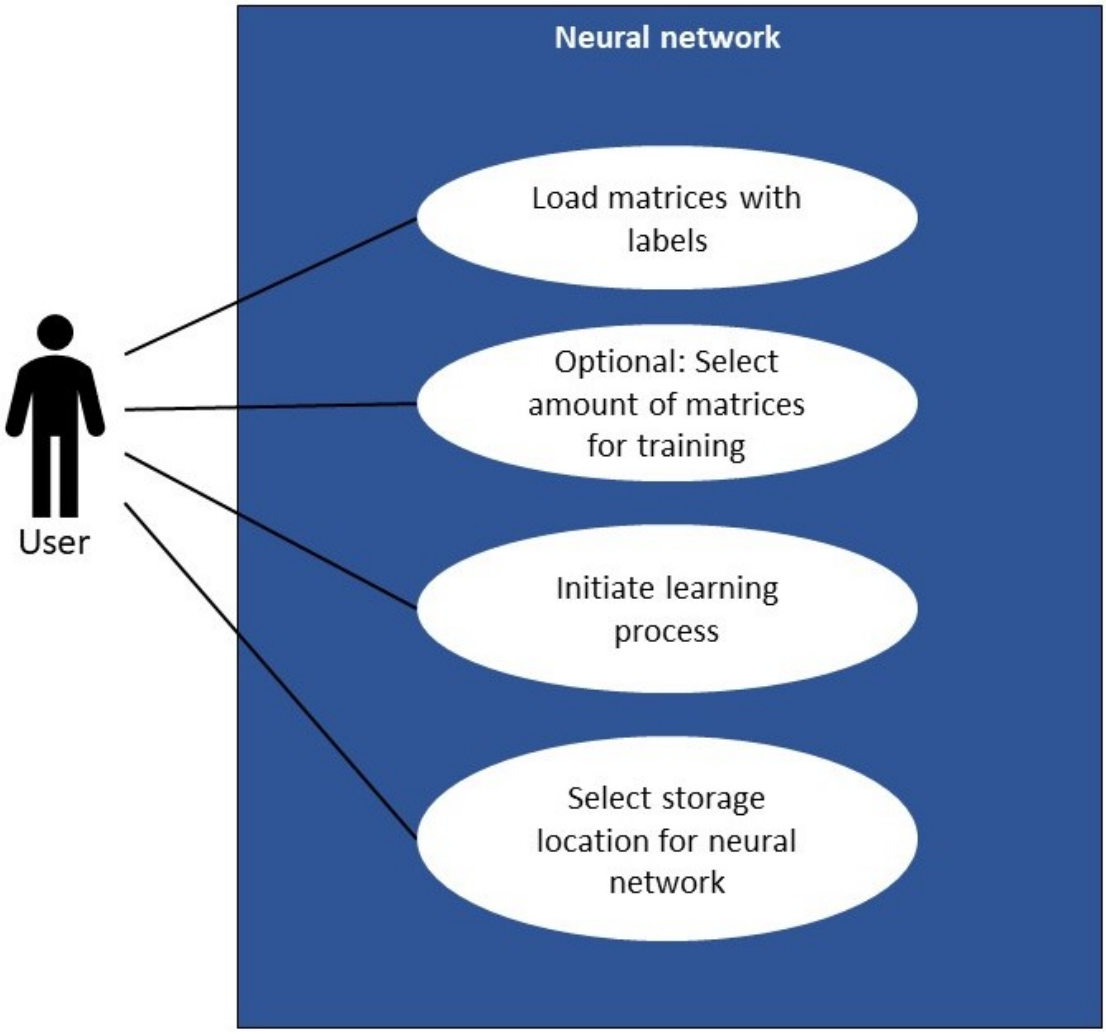
8.2.1 Matrix collector



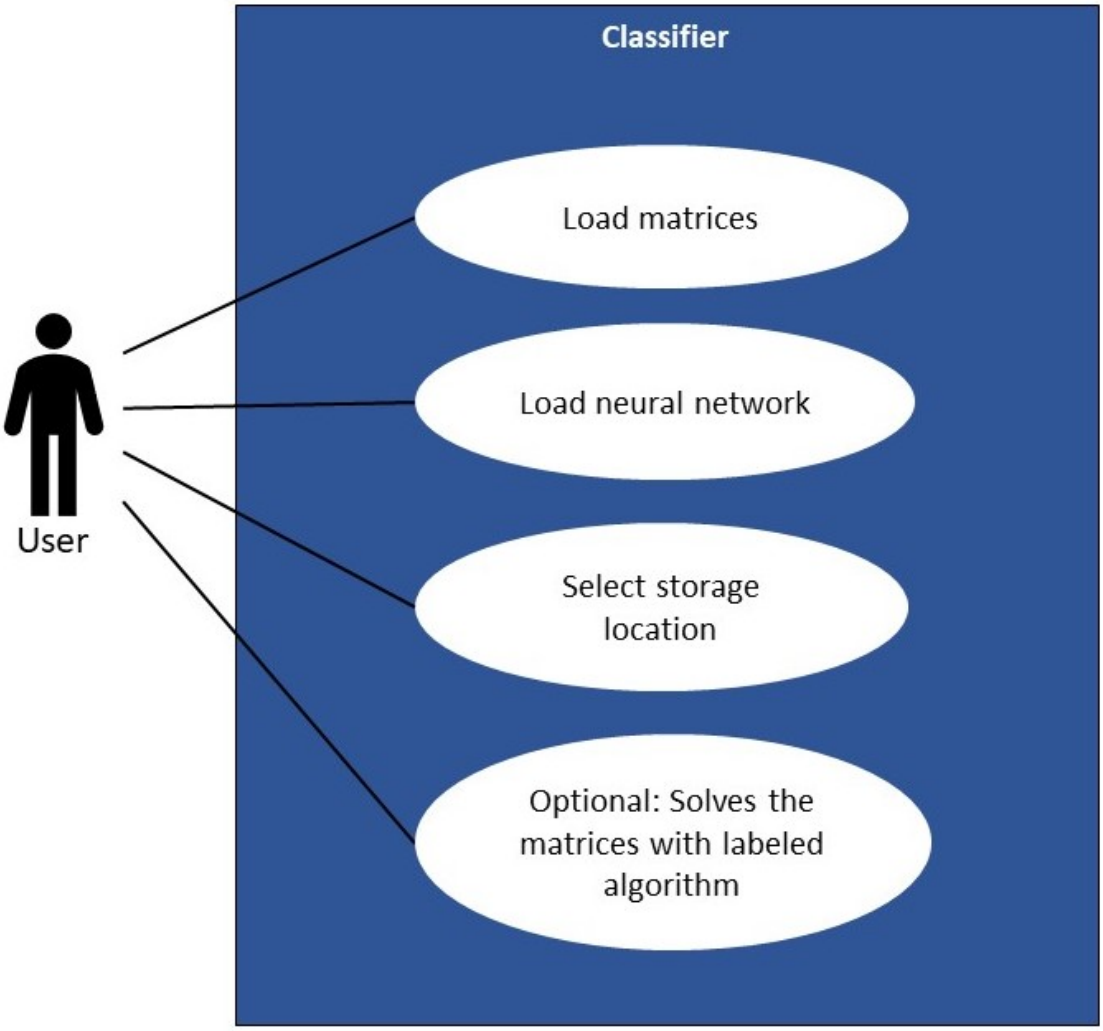
8.2.2 Matrix labeler



8.2.3 Neural network



8.2.4 Classifier



8.3 Command line options

- /B10/**collect -a <amount> -n <name> -s <size> -d <density> -p <path> -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. Default is 0.2
  - **-p <path>** (optional) Path where the created/downloaded matrices will be saved
  - **-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
  - Error, in case **-p <path>** is not a valid path

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

Arguments

- **label** (command) If set, enter labeling mode

- **-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

- **-s <saving path>** (optional) Path where 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

- Error, in case **-s <saving path>** is not a valid path

- /B30/**train -p <path> -n <name> -t <train> -s <saving path>**:
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 networks 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

- **-s <saving path>** (optional) Path where the neural network state will be saved

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
- Error, in case **-s <saving path>** is not a valid path

- /B40/**classify -p <path> -n <network> -s**:
  The user is able to pass a matrix to 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>** (optional) Path to the trained neural networks, if not set, uses the neural network shipped with the program
  - **-s** (optional) (flag) If set matrix will also be solved after classification.

  Print

  - The preconditioner/iterative solver combination which will solve the given matrix the fastest
  - (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

- (optional)/B50/**labeler list**:
  The user is able to retrieve a list of the available and used algorithms for the labeling module

Arguments

- **list** (command) If set, enter list mode

Print

- A list of all algorithms the labeling module currently uses and is able to use
- Error, in case the user passed more arguments

- (optional)/B60/**label add <algorithm>**:
The user is able to add an algorithm that will be used for labeling matrices.

  Arguments

  - **add** (command) If set, enter adding mode
  - <**algorithm**> A list of all algorithms the user wants to add, separated by blanks

  Print

  - A message if the adding worked
  - Error, in case the entered algorithm/algorithms is not supported
  - Error, in case other arguments were passed

- (optional)/B70/**label remove <algorithm>**:
The user is able to remove an algorithm from the list of used algorithms.

  Arguments

  - **remove**(Command) If set, enter removing mode
  - <**algorithm**>A list of algorithms the user wants to remove from the used algorithms

  Print

  - A message if the removing worked
  - Error, in case the entered algorithm/algorithms is not found.

- Error, in case other arguments were passed

9 Feasibility Study

9.1 Technical Feasibility

There exist programs that try to solve the same problem, but with different approaches. Most of them scan the matrices for special metrics and based on that decide which solver will be the fastest. So there are metrics the neural network might find, to predict the optimal solver for a given matrix. Other than that, most of the things needed can already be found in plugins like Keras for the neural network and ssget for downloading matrices from the Suite Sparse. So there definitely is a way to accomplish this task and the only question is how accurate the classifier will be.

9.2 Alternatives

As mentioned before, there are programs that try to solve the same problem and some of them are also quite accurate at it. But for the moment nobody can tell if our approach might be more accurate.

9.3 Personal Feasibility

The team consists of five trained computer scientists with some of them already having experience regarding machine learning techniques. There are also two experts that can be contacted when the team needs some help with more difficult questions.

9.4 Legal Feasibility

Since the team uses the 2-Clause BSD License for their written software, there is legal feasibility because the team holds the sole copyright to their project but is not responsible for any consequential damage, caused by the use of their software.

10 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.

classifier The classifier is 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 training module.

collector The collector is the first module in the program. Responsible for generating artificial matrices and collection preexisting matrices from the suite sparse matrix collection.

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.

gaussian noise The gaussian noise is statistical noise having a probability density function equal to that of the normal distribution, which is also known as the Gaussian distribution. In other words, the values that the noise can take on are Gaussian-distributed.

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

GPU A GPU is a graphic processing unit which has specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.

grayscale sparsity pattern image Grayscale sparsity pattern image is an image, displaying the zero and nonzero areas of a matrix, by covering the zero areas white and all other areas by a shade of gray, depending on the value.

iterative solver In computational mathematics, an iterative solver does a mathematical procedure that uses an initial guess to generate a sequence of improving approxi-

mate solutions for a class of problems, in which the n-th approximation is derived from the previous ones.

labeling module The labeling module is the second module in the program. Responsible for executing a given set of matrices with all the preconditioner/iterative solvers combination specified. It will furthermore label each matrix with the fastest combination.

Linux Linux is an open-source software operating systems.

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.

preconditioner In mathematics, preconditioning is the application of a transformation, that conditions a given problem into a form that is more suitable for numerical solving methods.

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.

Suite Sparse Suite Sparse is a suite of sparse matrix algorithms and Java interface to the Suite Sparse Matrix Collection.

training module The training module is the third module in the program. Responsible for training a deep neural network with the set of matrices and labels given by the labeling module.

Windows Microsoft Windows is a group of several graphical operating system families, all of which are developed, marketed, and sold by Microsoft.