Karlsruher Institut für Technologie

Design Document

Numerical Linear Algebra meets Machine Learning

Fabian Koffer

Simon Hanselmann

Yannick Funk

Dennis Leon Grötzinger

Anna Katharina Ricker

Supervisors

Hartwig Anzt Makus Götz

December 13, 2018

Contents

1 Module Interaction

1.1 Class Descriptions

1.1.1 Class CommandLineInterface

The view represents the concrete command line interface. Therefore it only consists of two methods. The first one is readInput that receives a message that will be displayed and reads the next user input. The other method is createOutput. This method prints a string to the CLI.

1.1.2 Interface OutputService

The OutputService interfaces can be implemented and passed to a module to receive the output of the modules. Therefore it has methods that represent different ways output can be displayed.

1.1.3 Interface Subscriber

The Subscriber interface only provides the method update() which will be triggered by an Observable upon receiving new values.

1.1.4 Class Observable

The Observable class can be used to notify Subscribers when new values are provided. Subscribers can subscribe themselves to an Observer to be notified get notifications. The next() method calls update() on each subscriber.

1.2 Class CLIOutputService

This class implements the OutputService and the Subscriber interface. On creation it gets a reference of the View to which it will pass the lines the modules wants to output. It also implements the Subscriber interface to subscribe itself to an observable. This can be used to display lines that are overwritten with new values like an progress bar or a counter.

1.2.1 Class Controller

The controller is the main entry point for the program execution. It creates the view, receives the user input, calls the parser to create a command from the input and starts the module the user wants.

1.2.2 Class CommandParser

The CommandParser is a static class that gets the input which the user entered and parses it to a concrete command-object.

1.2.3 Class Command

The Command class holds all the information entered by the user that is needed to execute a module. There is one command subclass for each module and the command class also validates that all parameters are available to run the module. The command also has a execute method which runs the specific module with all the arguments it needs.

1.3 Activity Diagrams

1.4 Class Diagrams

1.5 Sequence Diagrams

2 Collector

2.1 Class Description

2.1.1 Class Collector

The Collector class is responsible for collecting a given amount of matrices and saving it into a HDF5 dataset. When the user types collect into the CLI, a collector Object will be created and the public method collect() with its parameters: amount, name, size, density and path will be called. The class has a Saver class attribute and a Generator class

attribute. It uses methods from the Generator class to get matrices to collect and methods from the Saver class to save the collected dataset. (see the collect method Activity Diagram for a more detailed overview). The Collector class is the interface between matrix collecting and the CLI and conceals all the classes of the Collector described in the following.

2.1.2 Class Saver

The Saver class is just responsible for saving a given matrix dataset. Its only method is the save(dataset, name, path) method, which is called by the collect method from an Collector object. The save method takes an NPArray as a matrix dataset, converts it into an HDF5 file and saves it into a given directory with a given name.

2.1.3 Class Generator

The Generator class is responsible for actually generating matrices by transforming raw matrices from SuiteSparse and validating them. The generate(size, density):Matrix method is called by a Collector object, uses the Matrix class to initialize an empty matrix, uses the Ssget class to fetch and transform matrices from the SuiteSparse collection and uses the static Validator.validate method to check if the matrix is regular and can be returned.

2.1.4 Class Ssget

The Ssget class is responsible for fetching matrices from the SuiteSparse collection, transforming them and returning them. Its getMatrix method is called by a generator object. The getMatrix method uses the Matrix class to initialize a matrix, then the private downloadMatrix method to fetch a matrix from SuiteSparse, and after that uses its private cutMatrix method to cut a fixed size, regular matrix out of it.

2.1.5 Class Validator

The Validator class is a util class and responsible for validating given matrices(checking for regularity) Its only static method validate takes a matrix and returns true for regular, and false for not regular.

2.2 Activity Diagrams

2.3 Class Diagrams

2.4 Sequence Diagrams

3 Labeling Module

3.1 Class Description

4 Training Moulde

4.1 Class Description

4.1.1 Class Configuration File

The configuration file is a text file. It is used to specify all necessary information the class neural network needs to train the neural network. If the user does not change anything in the configuration file, defaut options will be used.The configuration file is organized in four main categories.

1. loading path of the set of matrices
2. saving path for the neural network
3. loading path for the neural network
4. model definition and hyperparameters abc

The loading path of the set of matrices is the path in which the matrices that are used for the training and testing are stored. The training module only supports one hd5 file. If the path is any other file, the labling module will print an error(would crashing make sense if the user has to change the config file anyway?). For the training and testing making sense there should be at least 500 matrices in the hd5 file. Otherwise the accuracy of the neural network will be so low that i can not be used for classification. If there is no path specified, the training module will use a default path. In the default path will be the latest matrices that the labling module has produced.

The saving path for the neural network is the path where the trained and tested neural network will be safed. It will be safed as a Keras model. If there is no path specified, the neural network will be safed at a default destination. If there is no path for the neural network specified in the module Classifier the module will use this default path to load its neural network.

The loading path for the neural network is strictly optional. If this path is specified the training module will use the neural network in the path for training and testing. This option enables the user to use a pre-trained neural network for training. This could be the case if the user interrupts the training process at a certain time and wants to to repeat the training later. Other use cases are of course possible too. The neural network has to be a model of the Keras framework. If the path is any other file the training module will print an error(crash?). If this path is not specified the training module will create a new neural network(with the model definition and hyperparamters of the next category) and train with it.

The model definition and hyperparameters are used to determine which neural network will be trained and tested. The model definition determines the following:

- the amount of layers
- the amount of nodes in every layer
- the kind of neural network(e.g. Convolutional)
- the activation function
- the regularization

The hyperparamters determine the following:

- the dropout
- the batch size
- how much of the data should be training and how much should be testing data

4.1.2 Class TrainingModule

The TrainingModule class is responsible for the training and testing of a neural network. It can not be instantiated, since it is a utility class.The structure is mainly oriented towards the keras workflow and will be further described later. The class offers one public method, the method train().
When the user types train() in the CLI the method train in the class TrainingModule will be executed in the following manner(see the activity diagram for a graphical overview).

- load the configuration file
- load the matrices
- seperate matrices in training and test data
- train a preexisting neural network or a new one(depending on the configuration file)
- test the neural network
- save the neural network

The configuration file that gets loaded will be used to specify the subsequent points.

The configuration file will determine from which path the labeled matrices will be loaded. If there were no changes made in the configuration file, the default path will be used(see the class description of the configuration file). The labeled matrices will be loaded in one hd5 file. If the path links to any other file, the class TrainingModule will print an error to the command line (crash?).

After that the class TrainingModule will seperate the training and test data. How the data will be seperated is specified in the configuration file.

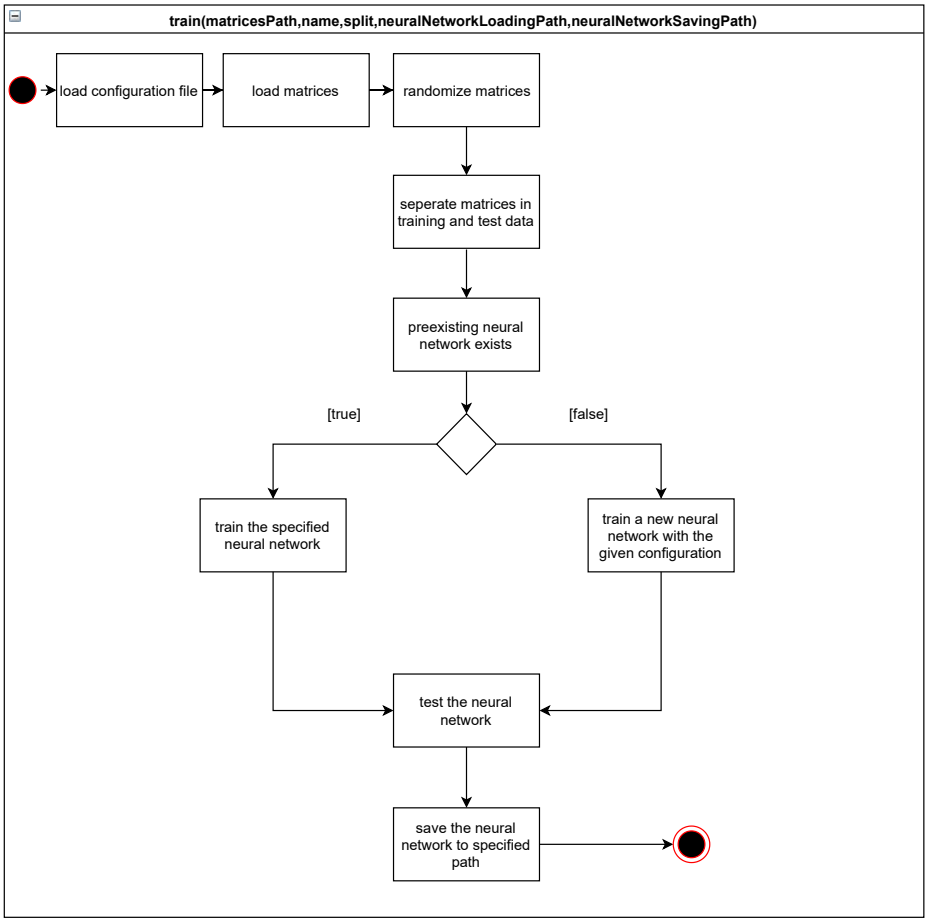
Following there are two alternatives If the user has specified a neural network in the configuration file, the class TrainingModule will train this neural network with the labeled matrices for the training. If the user has not specified a neural network in the configuration file, the class TrainingModule will create a new neural network with the specifications in the configuration file. If there are no model definitions in the configuration file the class TrainingModule will use the default neural network(see default neural network). The class TrainingModule then proceeds with training the new neural network

with the labeled matrices for the training. In both cases the current loss will be continously printed to the command line.

Now the neural network is trained. The class TrainingModule proceeds with testing the neural network with the labeled matrices for the testing. This process will determine the accuracy of the neural network on the given test matrices. The accuracy will be printed on the command line.

After that the neural network will be safed as a keras model. The path for the saving is specified in the configuration file.

4.2 Activity Diagrams


train(matricesPath,name,split,neuralNetworkLoadingPath,neuralNetworkSavingPath)
load configuration file
load matrices
randomize matrices
seperate matrices in training and test data
preexisting neural network exists
[true]
[false]
train the specified neural network
train a new neural network with the given configuration
test the neural network
save the neural network to specified path

4.3 Class Diagrams

4.4 Sequence Diagrams

5 Classifier

5.1 Class Descriptions

5.1.1 Class Classifier

5.1.2 Interface Algorithm

This interface is for the different algorithms for solving a given matrix.

5.1.3 Class ConcreteAlgorithm

The Concrete Algorithm class is for solving a matrix in a certain way. This means a certain approach to solve a matrix.

5.1.4 Class Matrix

5.2 Class Loader

5.3 Class Validator

5.3.1 Class Neural Network

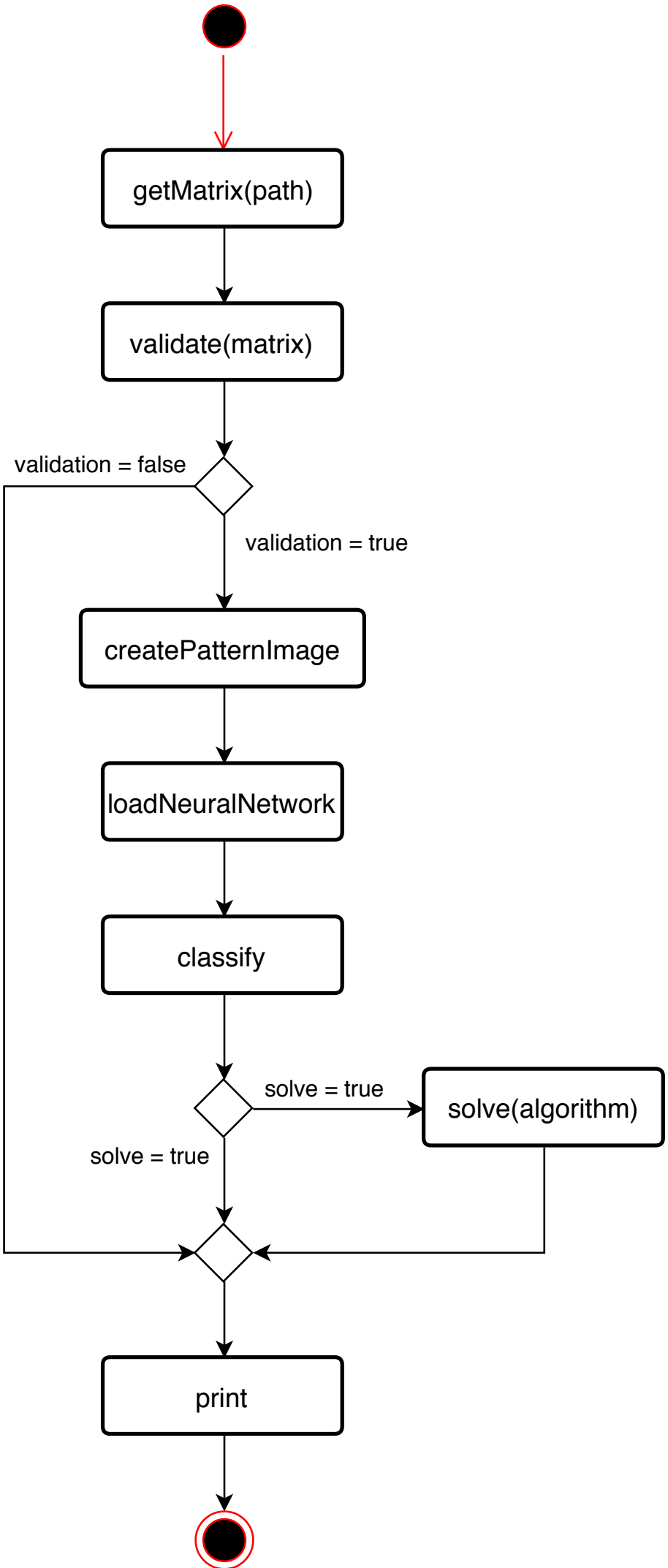
5.3.2 Class PatternImageCreator

This class creates a Grayscale Sparcity Pattern Image out of a given matrix.

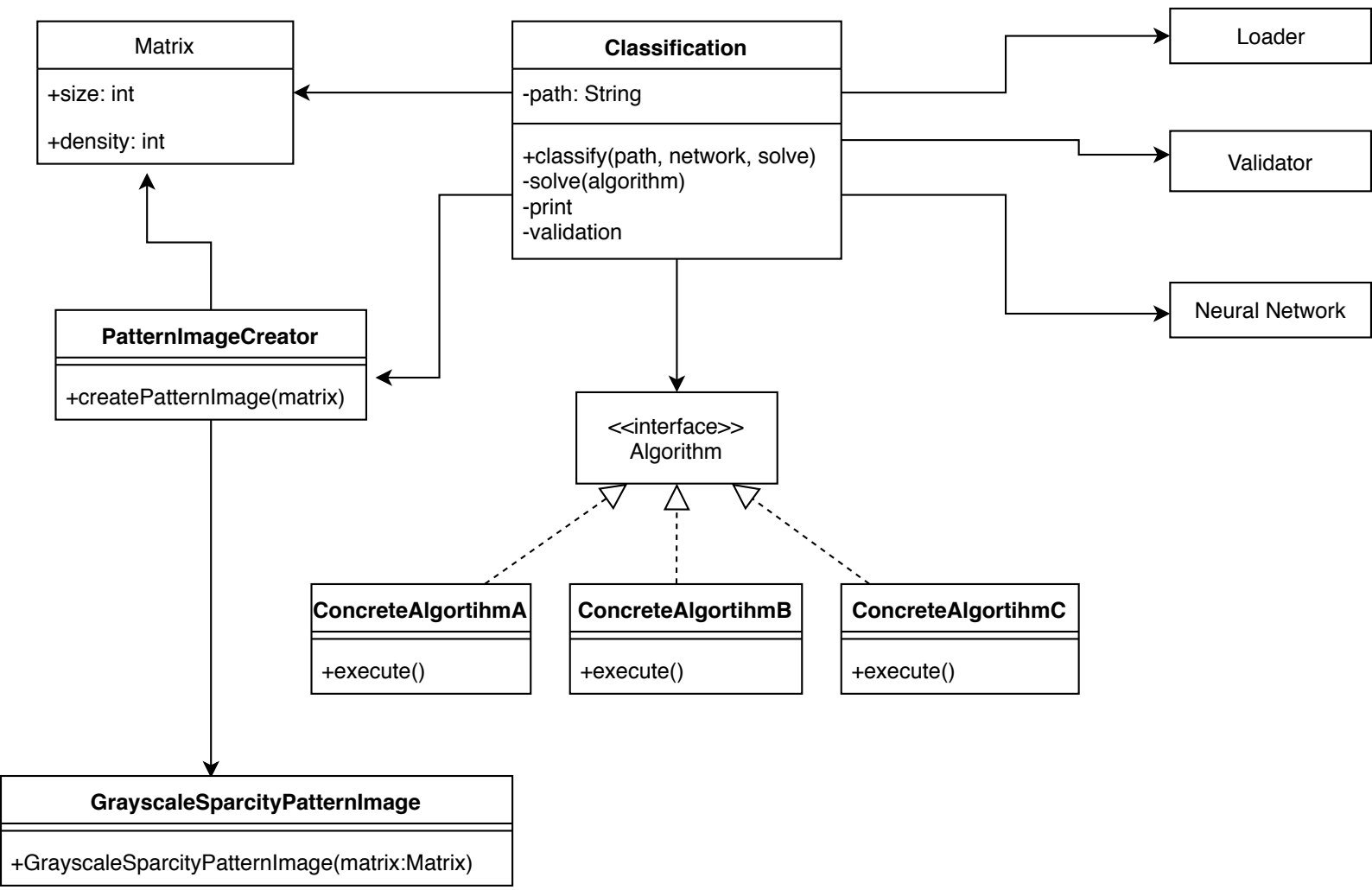
5.3.3 Class GrayscaleSparcityPatternImage

5.4 Activity Diagrams

Classifier.classify(path, network, solve)



5.5 Class Diagrams


Matrix
+size: int
+density: int
Classification
-path: String
+classify(path, network, solve)
-solve(algorithm)
-print
-validation
Loader
Validator
Neural Network
PatternImageCreator
+createPatternImage(matrix)
<<interface>>
Algorithm
ConcreteAlgortihmA
+execute()
ConcreteAlgortihmB
+execute()
ConcreteAlgortihmC
+execute()
GrayscaleSparcityPatternImage
+GrayscaleSparcityPatternImage(matrix:Matrix)

5.6 Sequence Diagrams

6 Explanations

6.1 default neural network

how is the nn structured(layers,activation function), what is it trying to achieve,...

7 Sequence diagrams

8 Glossary