Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Bau- und Infrastrukturmanagement
Infrastrukturmanagementgruppe
ETH Zürich
Prof. Dr. Bryan T. Adey

Projektarbeit Master-Studium FS 2014

Development of Neural Network Model

Autor: Jonathan Schneibel, ETH-Nr. 10-925-006

Betreuende: Prof. Dr. Bryan T. Adey
Borja García de Soto Lastra

Zürich, den 30.05.2014

ETH Zürich
Institut für Bau- und Infrastrukturmanagement
Professur für Infrastrukturmanagement
Professor Dr. Bryan T. Adey

1 Summary

Neural network models (NNs) are a technology that emerged in recent decades and found widespread use in many industries, including project and infrastructure management. In this work, a literature reviews goes over the concepts of how NNs work and how they can be applied, followed by the development of a practical NN model. The chosen model is a multilayer feedforward network using sigmoid transfer functions and applied to the problem of estimating the quantities of construction materials of tall steel-framed buildings, divided into the upper structure and the foundation. The implemented model using MatLab and Excel relies on mass-producing and evaluating NNs to find the best parameters necessary to generate the final NNs used for estimating the materials. Finally, the NNs are analysed and discussed. The importance values of the input variables are calculated and used as importance weights for a case-based reasoning model. In comparison, the developed NN model was found to perform better than the case-based reasoning model in this application.

2 List of Contents

3 List of Figures

4 List of Tables

5 Acknowledgements

This project work is part of the ETH master's curriculum in civil engineering and in this respect I want to thank Prof. Dr. Bryan T. Adey for offering the opportunity to work on an interesting topic which isn't taught in-depth in any of the lectures held at D-BAUG. I also want to thank Borja García de Soto Lastra for supervising the project and offering immediate and competent advice whenever needed.

6 Introduction

Neural network models (NNs) have found increasingly widespread use in many fields of work over the last few decades. One of the fields that can profit from NNs is project and infrastructure management. To show the capabilities of NNs in this area, a review of relevant literature as well as an example NN used to estimate the quantity of materials required in the construction of tall steel-framed buildings will be presented.

The literature review can be found in chapter 7 and will explain what NNs are, how they work and how they can be applied. A comparison to traditional estimation methods used in project and infrastructure management will also be made. Due to the large number of NN paradigms of varying complexity available, the scope of this work will be limited to multilayer feedforward networks using sigmoid transfer functions.

The insight gained from the literature review will then be used in chapter 8 to create a set of rules used to heuristically guide the creation, application and evaluation of a NN.

Finally, the framework will be applied in an example to estimate the quantity of materials required in the construction of tall steel-framed buildings in chapter 9. The calculations are done in MatLab and stored in Excel files using the data provided by the supervisors, which can be found on the USB drive (see appendix 13.1). The results of the implementation are discussed in chapter 10.

7 Literature Review

The mathematical concept of NNs is inspired by the mammalian brain, which can be simplified as consisting of processing elements called neurons and the connecting elements between them called axons, synapses and dendrites [Bishop 1994]. An electrical signal from a neuron will be sent through its axon to a synapse where it will be transferred to the dendrites of another neuron. The receiving neuron will then process the signal and as a result will either send a signal of its own (along its axon to other neurons) or not. In the theory of NNs and in this work, the element corresponding to the neuron is referred to as processing element or *node*, while the synapse is the *connection* between two nodes. The work that a single node does is to sum all the inputs it receives from preceding nodes, calculate the output according to a mathematical *activation function* and then send the output to the following nodes it's connected to.

Analogously to the brain, NNs are constructed so that they can solve complex problems and learn from experience [Bishop 1994]. Using a set of training data as input with known output, the parameters of the NN can be adjusted to generalize the data and estimate the outputs of new inputs accordingly. In order to effectively and efficiently adjust the parameters, a learning algorithm as well as some experimentation is required. This is usually the most time-consuming part of creating a practical NN.

7.1 The Multilayer Perceptron

Different network types have been developed differing mainly by *topology*, i.e. how many nodes there are and how they are connected to each other. The most commonly used types being the so-called radial basis function networks and multilayer perceptrons or multilayer feedforward networks [Bishop 1994]. Here, we will focus on the latter using sigmoid activation functions. They are characterized by their layered structure as visualized in Figure 1 and by the nodes processing signals with a sigmoid activation function (e.g. logistic or hyperbolic tangent function) [Bishop 1994]. Each *layer* consists of a varying number of nodes which are all connected to all nodes of the next layer. In addition, each layer has a node called *bias* which has a constant output of 1. Attached to each connection is a *weight* used to linearly scale signals before they are summed and then processed by the receiving node.

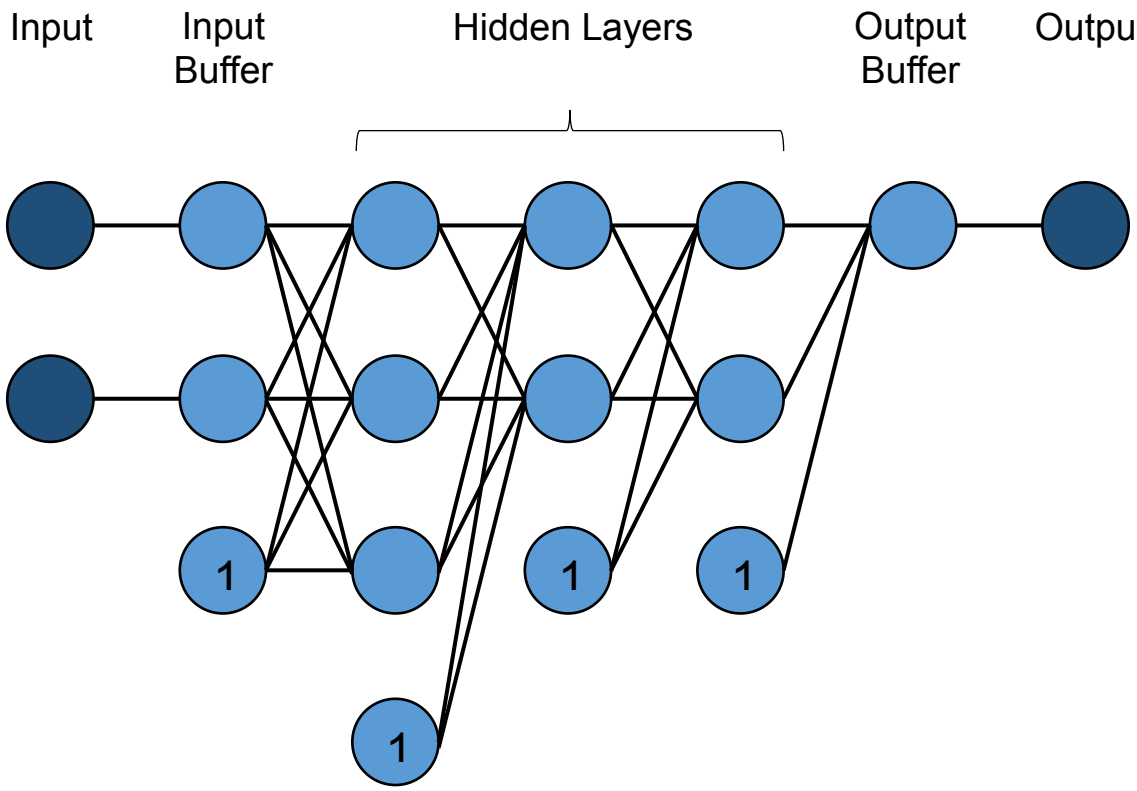

Figure 1: Example structure and terminology of a multilayer feedforward network.

The first and last layers are called the input and output *buffer* and are used to scale the inputs and outputs to a given range (for example to [-1,1] and back to the original range). The other layers are referred to as *hidden layers* processing the inputs according to the activation function of each node and the weights attached to the connections. By changing the weights according to a learning algorithm, the desired mapping from the input to the output can be achieved. In fact, it is possible to map any continuous function to arbitrary accuracy using the multilayer perceptron [Bishop 1994].

7.2 Learning Algorithms

Different algorithms exist to find the optimal parameters of a given NN. In this chapter, we focus on the learning algorithms called the generalized delta rule and genetic algorithms, both of which can be applied to the multilayer perceptron. They represent two very different approaches to optimizing a network and have their respective advantages and disadvantages. Both are called supervised, which means that the desired outputs of the training inputs are known. The goal of the learning algorithm is to minimize the difference between actual and

desired outputs, often represented by the mean square error of the outputs [Boussabaine 1996].

In both approaches, the available data is split into a training set and a testing set. The learning algorithm is used on the training set and the resulting network parameters will then be validated using the testing set [Boussabaine 1996]. This is because good performance on the training set does not strictly imply good performance on new data, like the testing set. It is possible that the NN has too many nodes which leads to overfitting, i.e. the NN learns exactly how the training inputs are mapped to the desired outputs without extracting the essential relationships between the variables necessary to predict the outputs of new data. Vice versa, too few nodes lead to bad generalization due to oversimplification. Changing the network configuration in order to improve test performance requires some experimentation but can be guided by a set of rules [Hegazy 1994].

7.2.1 The Generalized Delta Rule

The generalized delta rule is a gradient descent method used to minimize the sum-of-squares error function of the training outputs. The optimization happens iteratively, with each iteration moving a certain distance in the direction of the negative error gradient in weight space, which is the direction of the highest decrease of the error function. Thus, the algorithm will find a set of weights that causes a low error on the training set [Bishop 1994].

A disadvantage of this approach is that the algorithm might end up in a local minimum of the error function instead of the desired global minimum. This can be counteracted by increasing the learning rate (determined through experimentation), which represents the distance travelled in weight space with each iteration. A high learning rate will also reduce training time but can result in instability of the algorithm, depending on the curvature of the error function [Bishop 1994]. Another way to evade local minima is to randomize the set of weights used at the beginning of the algorithm. Different realizations of the initial random weights will lead to different ‘descents’ on the error surface. Running the algorithm several times with newly randomized weights will produce some NNs that have been trapped in a local minimum and some that have reached an error new the global minimum.

Another term besides the learning rate that can be introduced to speed up the training process is the learning momentum. If a component of the error gradient doesn’t change its sign in successive iterations, the learning momentum will cause this component of the error gradient

to increase and therefore the algorithm will reach a minimum in fewer iterations (as larger values of the gradient result in larger changes the of weights) [Bishop 1994].

7.2.2 Genetic Algorithms

The approach of genetic algorithms to optimization is completely different to the generalized delta rule. Genetic algorithms are inspired by biological evolution where a group of individuals (a population) reproduces and slowly evolves by preserving and improving the well-performing genes, which have been created by random changes, in their offspring [Melanie 1999].

The three main operations of a genetic algorithm are selection, crossover and mutation. During selection, the properties (genes) of all the NNs are evaluated and the well-performing ones are selected for reproduction. During crossover, the selected properties are combined to create new ones which hopefully perform even better. Finally, mutation randomly changes some properties, making sure that the search space of the properties isn't limited by the initial properties. Iterating this process for many iterations (generations) creates individuals that will eventually perform better than the initial population [Melanie 1999].

7.3 Creating a Neural Network Model

Using the general procedure proposed by Hegazy [Hegazy 1994], the development process consists of four main phases *concept*, *design*, *implementation* and *recall*. During the concept phase, the paradigm is selected (e.g. the multilayer feedforward network) depending on the given problem. In the design phase, the problem is analysed to determine the data required and how to acquire it. In addition, the problem is structured depending on how many inputs and outputs there are, their data types and constraints regarding the acquisition of training data. Next, the application is implemented, starting with the acquisition and preparation of the data and continuing with the coding of the programs required to perform all calculations. Using the programs and data, the network is created, trained and tested until a satisfactory result is achieved, often including a lot of experimentation and intuition. The recall phase contains the actual usage of the produced NN.

7.4 Use of Neural Network Models

The applicability and performance of NNs in practical situations has been subject of some research, especially their performance in comparison to other methods [Boussabaine 1996]

[Olden 2002] [Kim 2004]. In the fields of construction, project and infrastructure management, NNs can be used for construction cost estimation [Hegazy 1998] [Kim 2004], sequencing of construction tasks [Boussabaine 1996] or the estimation of construction material quantities as presented in chapter 9. NNs excel in situations where ample data for network training can be provided, mechanistic or analytic models are hard to develop, and the models need to handle noisy data [Bishop 1994].

On one hand, NNs represent a powerful tool which can be used for complex problems like pattern recognition, estimation and classification where traditional methods often fail due to their inability to adapt to new situations or handle non-linearity [Olden 2002] [Hegazy 1994]. On the other hand, NNs are called black boxes because they don't inherently offer any insight into why a specific output has been produced or how the input variables relate to each other [Olden 2002]. Their lack of explanation regarding the underlying mechanics of the problems can lead to uncomfortable situations in decision making when the outputs are in conflict with traditional theories or expert opinion [Boussabaine 1996]. As a result, various techniques to interpret neural networks have been developed and evaluated [Olden 2002] [Olden 2004].

7.5 Interpreting a Neural Network Model

In order to understand why a NN delivers a specific output or to gain some insight into the real-world processes governing the modelled environment, it is necessary find a suitable technique for interpreting the weights of a NN as the weights encode the knowledge the network gained during training. Olden investigated nine different methods for ranking the variable importance in NNs using simulated data where the exact relationship between the different inputs is known [Olden 2004]. It has been found that the connection weight approach as applied in chapter 10.2 is a simple yet accurate and precise method for interpreting NN models. Other calculations such as the partial derivatives and input perturbation produce modest results [Olden 2004].

8 Formulation of the Proposed Model

The proposed NN model is a multilayer perceptron using the generalized delta rule as learning algorithm. In the following subsections, the procedure chosen to find the best performing NN configuration to solve the given problem is outlined and some of the reasoning behind it explained.

8.1 Overview

The general tasks involved in finding the best performing NN configuration consist of *A) data analysis and preparation, B) generating possible NN configurations* and *C) selecting the best performing NN configuration* as shown in Figure 2:

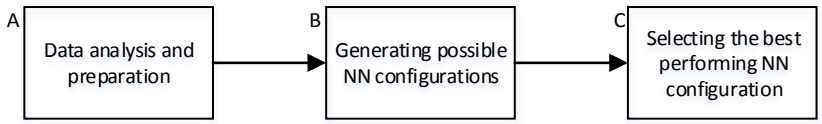

Figure 2: Overview of the development process.

Data analysis is necessary to determine the input and output variables of the NN as well as their data types. With the inputs and outputs defined, a multitude of NN configurations with varying parameters (number of layers, number of nodes, activation function, etc.) are generated. The generated NNs are evaluated to determine their individual performance and the best performing NN can then be used to predict the outputs of new cases.

8.2 Data Analysis and Preparation

First, the input and output variables of the provided data set to be used in the NN have to be chosen. Using more relevant input variables will generally provide a more accurate model, but this comes at a computational cost which cannot be neglected in the chosen procedure. Therefore, the user might have to decide between two options (to include certain input variables or not) depending mainly on the accuracy requirement of the final NN model and the available computational resources. Both can be hard to estimate and will require some experimentation and intuition to decide upon. Some tools which can assist in this process are outlined in the following subsection.

The output variables are determined by the problem definition and don’t require further preparation. It’s possible to change individual fields to improve readability or to remove them

if they show clear relationships between each other. However, this would only be necessary if there is a large number of output variables, which is usually not the case.

8.2.1 Removing Redundant Data Fields

If several data fields represent the same information, only one should be chosen as input variable. This is the case if there exist bijective functions between the data fields, theoretically allowing the user to replicate the data of all concerned fields if the data of one field and the bijective function is given. Which field to choose as input variable is an arbitrary decision. An example of this is shown in Table 1: The fields 'Storeys' and 'Height' contain the same amount of information, therefore it is sufficient to use only one field.

ID	Storeys	Height
1	3	15
2	5	23
3	5	23
4	4	19

→

ID	Storeys
1	3
2	5
3	5
4	4

Table 1: Data with bijective data fields.

Surjective relationships between data fields also imply redundancy. The field representing the image of the surjective function can be removed since the information it contains is only a subset of the information contained in the field representing the domain. In Table 2, an example is made using different customers of which each owns a certain number of buildings.

ID	Customer	Building count
1	Customer A	2
2	Customer B	5
3	Customer C	2
4	Customer D	9

→

ID	Customer
1	Customer A
2	Customer B
3	Customer C
4	Customer D

Table 2: Data with surjective data fields.

Another possibility is that there exists neither a bijective nor surjective relationship between two fields, but they are strongly correlated. This can be determined using the formula for the absolute value of the sample correlation coefficient between fields X and Y:

$$|r_{XY}| = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where $\bar{x}$ and $\bar{y}$ are the sample means and n is the number of cases in the data set. The range of $|r_{XY}|$ is 0 to 1, 0 indicating no correlation and 1 indicating a direct linear relationship between the fields. If the value is close to 1, the two fields combined don't contain much more

information than each field alone. What is exactly considered ‘close to 1’ and which field to use as input variable has to be determined by the user.

8.2.2 Data Types of the Input Variables

The inputs are further processed depending on their data type in the provided data set. If a field contains a large range of integers or rational numbers, they will be left as is. Fields with a limited set of possible values as inputs (i.e. categories) will be split into binary fields of all possible combinations of categories (as suggested in [Hegazy 1998]). As a result, each case will show 1 for its specific combination of categories and 0 for the others, as demonstrated in an example in Table 3: The field ‘Building type’ only accepts values ‘Apartments’ or ‘Offices’ and the field ‘Location’ only accepts values ‘Zürich’ or ‘Uster’, in the resulting fields abbreviated as ‘A’, ‘O’, ‘Z’ and ‘U’ respectively.

ID	Building type	Location
1	Apartments	Zürich
2	Offices	Uster
3	Apartments	Uster
4	Offices	Zürich

→

ID	AZ	AU	OZ	OU
1	1	0	0	0
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

Table 3: Splitting data fields containing categories into binary fields.

This kind of split is not necessary if the field contains a limited range of integers which directly represent a physical property of the examined object. As an example, if the number of lanes on a road network is in the integer range of 1 to 4, the data doesn’t have to be split into binary fields since the values are a direct physical measure of the roads. At the same time, it would be possible to handle them like categories, which isn’t recommended because this approach generally requires more computing power.

8.3 Generating Possible Neural Network Configurations

As could be seen in the previous subsection, the computational cost of various decisions have to be considered. This is because the chosen procedure heavily relies on ‘mass-producing’ optimized NN models, limited mainly by the computational resources available. This can be seen in the flowchart of Figure 3.

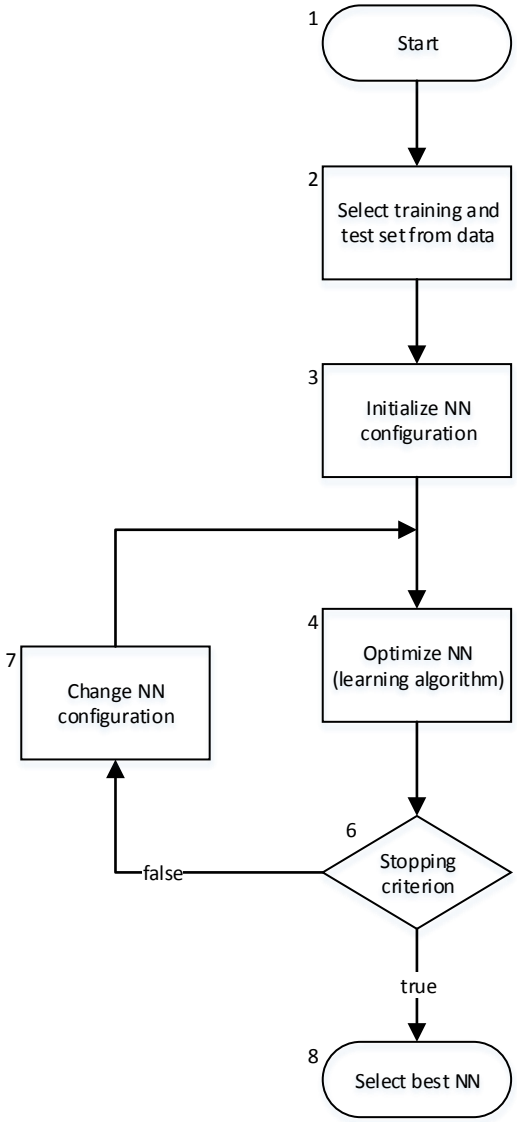

Figure 3: Algorithm to produce possible NN configurations.

In step *2*, the data is split into a training set and a test set. For this, a set number of data points is randomly selected for each building type (see chapter 8.2.2) and assigned to one of the two sets.

The stopping criterion *6* for the loop is that all possible different network configurations have been optimized. In this context, the term 'configuration' refers to the following parameters with their respective ranges:

- Number of hidden layers (one or two)
- Number of nodes (between one and twice the number of input variables per layer, based on [Hegazy 1994])
- Activation function for all layers (hyperbolic tangent or logistic)

For a NN with seven input variables, the first configuration to optimize would consist of a single node while the last configuration would consist of two layers with 14 nodes each, totalling to $14 \cdot (14 + 1) = 210$ different network topologies using the hyperbolic tangent as activation function and another 210 using the logistic function.

The hyperbolic tangent activation function is given by:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

And the logistic activation function is given by:

$$f(x) = \frac{1}{1 + e^{-a}}$$

It is apparent that the outputs of these two functions are in the range [-1,1] and [0,1]. The activation function is always chosen for all nodes of the NN.

For every NN configuration, the learning algorithm runs for a set number of iterations with certain learning parameters η (learning rate) and μ (learning momentum) and newly randomized initial weights. The performance of the resulting NNs is usually dependent on the values of η and μ, yet their optimal values cannot be easily determined for such a large number of different NN configurations. Therefore, automatically changing them every iteration to cover a broad range of η and μ will yield better results. Repeatedly generating new weights solves the problem of possible network paralysis. The flowchart of the learning algorithm (step *4* in Figure 3) is shown in Figure 4.

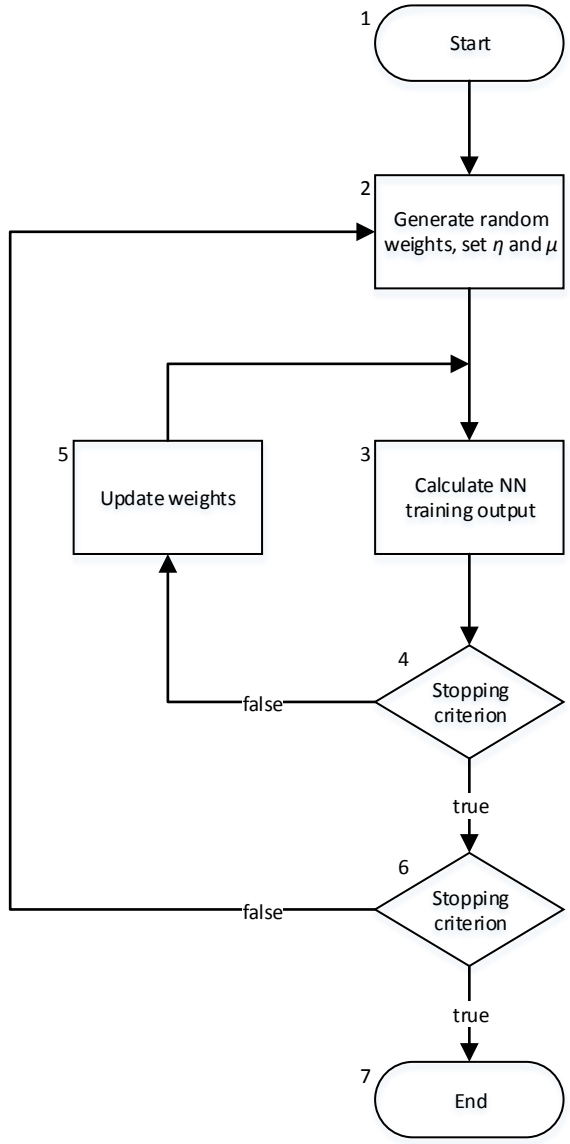

Figure 4: Learning algorithm used to optimize a given NN configuration.

Loop *3-4-5* is the actual generalized delta rule optimizing a given NN configuration, while loop *2-3-4-6* repeats the whole optimization with new initial weights and learning parameters.

The weight changes of the generalized delta rule in step *5* result in the lowest sum-of-squares error $\hat{E}$ which is defined as in [Bishop 1994]:

$$\hat{E} = \frac{1}{2}\sum_{n}\sum_{j}\left(X_{h,nj} - X_{h,nj,true}\right)^2$$

Where $X_{h,nj}$ is the output of the node of the output buffer (layer h) of data point n from the training set connected to output j. $X_{h,nj,true}$ is the actual output value provided in the training

set. The error function is further discussed at the end of chapter 9.2.1. The loops in Figure 4 are run for a set number of times determined by the user.

The results stored in the step *generating possible NN configurations* aren't the optimized weight matrices themselves. For each configuration (number of nodes/layers and activation function) the learning parameters η and μ used to find the lowest $\hat{E}_{test}$ and resulting training and test errors are recorded. This information will then be used in *selecting the best performing NN* to further improve a selection of the best NN configurations found in *generating possible NN configurations*.

As a summary, three nested loops characterize the algorithm used in *generating possible NN configurations*, listed from the innermost to the outermost loop:

- The *inner loop* updates the weights of a given NN configuration in order to lower the error $\hat{E}$. The number of iterations is set by the user.
- The *middle loop* changes the learning parameters η and μ and generates new random weights. The number of iterations is set by the user.
- The *outer loop* changes the network topology and activation function. The loop is iterated until the whole range of possible NN configurations has been exhausted.

The output of the algorithm is a list of all NN configurations with their respective lowest test errors $\hat{E}_{test}$ and the parameters used to reach those errors.

8.4 Selecting the Best Performing Neural Network

The performance of the NNs is assessed solely on the basis of the test error resulting from the outputs of the test data set. After calculating the test error for many different NN configurations in *generating possible NN configurations*, the best-performing configurations are taken and individually optimized again with manually set learning parameters. This step is less structured and will require more experimentation. The process is over when all selected NNs have been further improved and the weights of the best and final NN have been stored for future use.

9 Example Implementation

In this chapter, the proposed model from chapter 8 is applied to two data sets containing information about the upper structure and foundation of tall steel-framed buildings. The goal is to estimate the quantity of materials required in the construction of such buildings.

In order to find the best performing NN configuration, various programs have been created in MatLab R2012a to optimize and test as many different NN configurations as possible according to the steps *generating possible NN configurations* and *selecting the best performing NN configuration* in Figure 2. The program files can be found on the USB drive (see appendix 13.1). An overview over the intended workflow as well as a short guide on how to use the programs can be found in appendix 13.2.

9.1 Data Analysis

The two data sets about the upper structure and the foundation are completely separate and contain a different number of data points. Therefore, the two sets will be examined separately and two different NNs will be developed.

9.1.1 Upper Structure

There are 4506 data points available which are related to the upper structures. The following information is included in this data set, numbered from *1* to *13*:

1 Material	'HB' (hybrid steel/concrete) or 'RCC' (reinforced concrete), depending on the construction type of the structure.
2 Levels	7, 8 or 9, related to the number of storeys of the structure.
3 Stages	4, 5 or 6, related to the mechanical equipment found on different storeys of the structure.
4 Strings	1 or 2, related to the width of the structure.
5 Total height	Total height of the structure in meters.
6 Wind speed	Design wind speed in miles per second.
7 Ground acceleration	Design earthquake acceleration parameter in g (gravitational constant).
8 Soil bearing capacity	Bearing capacity of the soil in tons per square meter.

9 Soil factor Refers to the ground type according to design codes.

10 Concrete (columns) Amount of concrete in cubic meters used for the columns of the structure.

11 Reinforcement (columns) Amount of reinforcing steel in tons used for the columns of the structure.

12 Structural steel (slabs) Amount of structural steel in tons used for the slabs of the structure.

13 Formwork Total amount of framework in square meters used for the construction of the structure.

The last four fields contain the quantity of construction materials used and are therefore the output variables of the NN. The fields *1* to *9* are possible input variables and will now be analyzed and checked for redundancy according to chapter 8.2.

The coefficients of correlation between fields that are suspected to be strongly correlated have been calculated with the `CORREL` function in Excel and are shown in the symmetric Table 4:

r_{XY}	1	2	3	4	5	6	7	8	9
1	1								
2		1			0.878	0.009			
3			1		0.745	0.007			
4				1					
5		0.878	0.745		1	0.007			
6		0.009	0.007		0.007	1			
7							1		
8								1	-0.962
9								-0.962	1

Table 4: Some coefficients of correlation of the upper structure data set as calculated in Excel.

It is apparent that there exists no correlation between *6 Wind speed* and *2 Levels*, *3 Stages* and *5 Total height* which are directly or indirectly related to the height (and, one might suspect, to wind exposure), showing correlation coefficients of less than 0.01.

On the other hand, *8 Soil bearing capacity* and *9 Soil factor* are strongly correlated with a correlation coefficient of -0.962, indicating a nearly linear relationship between them. As a result, *9 Soil factor* has been removed from the data as it isn't necessary for the NN to perform well.

The relationships between *2 Levels*, *3 Stages*, *4 Strings* and *5 Total height* are more complicated. First, the coefficients of correlation of *2 Levels* and *3 Stages* with *5 Total height* are moderately high at 0.878 and 0.745 respectively. Second, there exists a surjective mapping from the combined fields *3 Stages*, *4 Strings* to *2 Levels*, as shown in Table 5:

Stages	Strings	Levels
4	1	7
4	2	7
5	1	8
5	2	8
6	1	8
6	2	9

Table 5: Surjective mapping from 'Stages' and 'Strings' to 'Levels'.

Therefore, if *3 Stages* and *4 Strings* are included as input variables, *2 Levels* can be excluded. In addition, due to the coefficient of correlation, *5 Total height* can be excluded as well. Another option would be to exclude *3 Stages* and *2 Levels* instead. In both variants, only one measure of height would be included in the data set. The decision was made to try both variants and see which one produces better NNs.

After splitting the fields of *1 Material* and *4 Strings* into binary fields according to chapter 8.2.2, the two variants of input and output data for the upper structure are as shown in Table 6, each counting nine input variables and four output variables:

Variant 1
Inputs
H1
H2
C1
C2
Total height
Wind speed
Ground acceleration
Soil bearing capacity
Outputs
Concrete (columns)
Reinforcement (columns)
Structural steel (slabs)
Formwork

Variant 2
Inputs
H1
H2
C1
C2
Stages
Wind speed
Ground acceleration
Soil bearing capacity
Outputs
Concrete (columns)
Reinforcement (columns)
Structural steel (slabs)
Formwork

Table 6: The two final variants of the upper structure data set after data analysis and preparation.

The fields *H1* to *C2* are binary fields where *H* and *C* mean that *1 Material* is ‘HB’ or ‘RCC’ respectively, and the digit refers to the number of strings.

9.1.2 Foundation

The data set concerning the foundations contains 526 data points. The following information is included in this data set, numbered from *1* to *11*:

1 Levels	7, 8 or 9, related to the number of storeys of the upper structure.
2 Stages	4, 5 or 6, related to the mechanical equipment found on different storeys of the upper structure.
3 Strings	1 or 2, related to the width of the upper structure.
4 Ground height	Height from the ground to the first storey measured in meters.
5 Total height	Total height of the upper structure in meters.
6 Wind speed	Design wind speed in miles per second.
7 Ground acceleration	Design earthquake acceleration parameter in g (gravitational constant).
8 Soil bearing capacity	Bearing capacity of the soil in tons per square meter.
9 Soil density	Density of the soil on the building site.
10 Concrete	Amount of concrete in cubic meters used for the foundation.
11 Reinforcement	Reinforcing steel used for the foundation according to code.

The last two fields contain the quantity of construction materials used and are therefore the output variables of the NN. The fields *1* to *9* are the possible input variables and are analyzed analogously to the data of the upper structure in the previous subsection.

The coefficients of correlation between fields that are suspected to be strongly correlated are shown in the symmetric Table 7:

r_{XY}	1	2	3	4	5	6	7	8	9
1	1			0.010	0.856	0.020			
2		1		0.000	0.711	0.039			
3			1						
4	0.010	0.000		1	0.400				
5	0.856	0.711		0.400	1	-0.011			
6	0.020	0.039			-0.011	1			
7							1		
8								1	0.986
9								0.986	1

Table 7: Some coefficients of correlation of the foundation data set as calculated in Excel.

The decisions are similar to those from the previous subsection: *9 Soil density* is removed in favour of *8 Soil bearing capacity* and *1 Levels* is removed in favour of one variant with *2 Stages* and one variant with *5 Total height*.

None of the fields are correlated to *4 Ground height*, which means that this field provides new information that wasn't available in the data set for the upper structures.

The two resulting variants for the foundation are shown in Table 8, each containing seven inputs and two outputs.

Variant 1
Inputs
Single string
Double string
Ground height
Total height
Wind speed
Ground acceleration
Soil bearing capacity
Outputs
Concrete
Reinforcement

Variant 2
Inputs
Single string
Double string
Ground height
Stages
Wind speed
Ground acceleration
Soil bearing capacity
Outputs
Concrete
Reinforcement

Table 8: The two final variants of the foundation data set after data analysis and preparation.

The fields *Single string* and *Double string* are binary fields with one of them showing *1* and the other *0* according to whether a structure has one or two strings.

9.2 Generating Possible Neural Network Configurations

The algorithm of Figure 3 has been implemented in the program `NN_GDA.m` using MatLab. First, the user inputs necessary to run the program are listed and major choices made when

designing the program are explained. Afterwards, the outputs of the program when using the previously created data sets for the upper structure and foundation are shown.

9.2.1 Program Inputs and Design Choices

The following list contains some of the important inputs that the program `NN_GDA.m` requires in order to run (a full list can be found in appendix 13.2.2):

- `input_filename`: The filename of the Excel file containing the data set, for example `'Input_Data.xlsx'`.
- `input_range`: The range of columns containing the input variables, for example `[1:8]` for columns 1 to 8.
- `input_range_type`: The range of columns containing binary data (the building types), for example `[1:4]` for columns 1 to 4.
- `output_range`: The range of columns containing the output variables, for example `[9:12]` for columns 9 to 12.
- `output_filename`: The output of the program will be written to a new Excel file with this name, for example `'NN_batch.xlsx'`.
- `activation_function`: The activation function to be used for all nodes, either `'tanh'` or `'logistic'`.
- `n`: The number of data points per building type to use for training, for example `100` will result in a total of 400 data points for the upper structure since there are four building types.
- `ntest`: The number of data points per building type to use for calculating the performance of an optimized NN configuration.
- `p_eta_start`: The upper limit to use for learning rate η in the first iteration of the generalized delta rule, for example `2`.
- `p_eta_end`: The lower limit to use for learning rate η in the first iteration of the generalized delta rule, for example `0.01`.
- `eta_ratio`: The ratio of learning rate η of the first iteration to that of the last iteration of the generalized delta rule, for example `10`.
- `mu`: The learning rate μ to use for the generalized delta rule, for example `0.9`.
- `NN_iterations`: The number of iterations where a given NN configuration is optimized (middle loop *2-3-4-6* in Figure 4), for example `100`.

- `GDA_iterations`: The number of iterations of the generalized delta rule (inner loop *3-4-5* in Figure 4), for example `400`.
- `max_nodes`: The maximum number of nodes in the two layers, for example `[5,8]` to generate $5 \cdot (8 + 1) = 45$ different NN configurations with a maximum of 5 nodes in the first layer and 8 nodes in the second layer. An empty matrix `[]` will set it to the default of two times the number of input variables for each layer.

Some of the parameters that had to be determined through experimentation and intuition when creating the code and setting the program inputs were the learning parameters η and μ and the number of iterations that the 'inner loop' and 'middle loop' (using the terminology from the summary at the end of chapter 8.3) run for.

For the learning momentum μ, a constant value of about 0.9 seemed to produce the best results for most used NN configurations, meaning that `mu=0.9`.

In regard to the learning rate η, lowering its value with every iteration of the inner loop produced good results. High values lead to large but imprecise weight updates (requiring few iterations to reach a minimum) while low values lead to small but precise weight updates (requiring many iterations to reach a minimum). Continually lowering η means that large weight updates are made in the beginning of the algorithm, when the minimum is still far away in weight space, and small weight updates are made at the end of the algorithm, when precision is more valuable than speed. With the number of iterations of the inner loop set to `GDA_iterations=400`, the actual values used for η are calculated as

```
eta = eta_start/(eta_start/eta_end)^((i-1)/(GDA_iterations-1));
```

where `i` is the current iteration of the inner loop and `eta_start` and `eta_end` are the values of η for the first and last iteration of the inner loop. The values of `eta_start` and `eta_end` are set again at the beginning of every iteration of the middle loop:

```
eta_start = p_eta_start/(p_eta_start/p_eta_end)^((p-1)/(NN_iterations-1));
eta_end = eta_start/eta_ratio;
```

The formulas include the current iteration `p` and total number of iterations `NN_iterations=100` of the middle loop. The other variables are inputs controlled by the

user and have been set to `p_eta_start=2` and `p_eta_end=0.01` (dictating `eta_start` for the first and last iteration of the middle loop) and the ratio between the first and last η of the inner loop being set by `eta_ratio=10`.

In summary, it is apparent that `eta` is lowered exponentially, starting at `eta_start`. With the chosen program inputs, `eta` is multiplied by the factor $10^{-1/(400-1)} \cong 0.9942$ with every iteration of the inner loop. The analogue is true for `eta_start`, starting at `p_eta_start` and being multiplied by $(2/0.01)^{-1/(100-1)} \cong 0.9479$ with every iteration of the middle loop.

Due to the fact that different activation functions result in a different range of output buffer outputs $\boldsymbol{X_h}$, error function presented in chapter 8.3 isn't suited to compare the performance of NNs with different activation functions. In order to find a suitable error function, the squared error $\hat{E}_{nj}$ of a single output *j* of a single data point *n* is expressed as a function of the final outputs $\boldsymbol{O}$

$$\hat{E}_{nj} = \left(X_{h,nj} - X_{h,nj,true}\right)^2 = (b-a)^2 \left(\frac{O_{nj} - O_{nj,true}}{O_{j,true,max} - O_{j,true,min}}\right)^2$$

where *a* and *b* denote the possible range *[a,b]* of $\boldsymbol{X_h}$ (a derivation of this equation can be found in appendix 13.5). Since the final outputs $\boldsymbol{O}$ and related errors are comparable between all NNs with the same output variables, defining a new error function *E* that is independent of *a* and *b* allows us to select the best performing NN from the NNs produced using both activation function 'tanh' (range [-1,1]) and 'logistic' (range [0,1]). The formula is further normalized by dividing by the number of test samples used, making sure that NNs tested with different a number of test samples can still be compared. The new error function E is calculated by dividing $\hat{E}$ by $n(b-a)^2/2$:

$$E = \frac{2\hat{E}}{n(b-a)^2} = \frac{1}{n}\sum_n \sum_j \left(\frac{O_{nj} - O_{nj,true}}{O_{j,true,max} - O_{j,true,min}}\right)^2$$

The rest of the algorithm doesn't have to be adjusted since there is a monotonic relationship between E and $\hat{E}$. Minimizing one error function will also minimize the other, therefore the generalized delta rule is still appropriate for optimization.

9.2.2 Program Outputs

Using the data sets of the upper structure and foundation with two variants each, a total of eight output files have been produced using the program `NN_GDA.m` with the 'tanh' and 'logistic' activation functions. The input parameters `p_eta_start` and `p_eta_end` have been set to 2.0 and 0.01 respectively for all files. The complete output files (which also include lists of the exact user inputs) can be found on the USB drive (see appendix 13.1).

The four best NNs of each data set and activation function (determined by the lowest E_{test}) and corresponding η and error values produced by the program `NN_GDA.m` are listed in Table 9 to Table 16.

Upper structure, variant 1, activation function ‚tanh'					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
16	0	1.450687	0.145069	0.006766	0.007568
14	16	0.211269	0.021127	0.009579	0.008136
12	0	0.065087	0.006509	0.007459	0.008222
14	12	0.248064	0.024806	0.009089	0.008824

Table 9: The four best NNs using the upper structure variant 1 data set and activation function 'tanh'.

Upper structure, variant 1, activation function ‚logistic'					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
16	0	2	0.2	0.005005	0.005446
9	0	2	0.2	0.005154	0.005575
14	11	2	0.2	0.005216	0.005594
13	0	1.53044	0.153044	0.005110	0.005601

Table 10: The four best NNs using the upper structure variant 1 data set and activation function 'logistic'.

Upper structure, variant 2, activation function ‚tanh'					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
15	0	0.471494	0.047149	0.008958	0.008267
8	14	0.380632	0.038063	0.008998	0.008643
13	0	0.423634	0.042363	0.008960	0.008758
5	11	0.997412	0.099741	0.010250	0.008830

Table 11: The four best NNs using the upper structure variant 2 data set and activation function 'tanh'.

Upper structure, variant 2, activation function ‚logistic'					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
5	15	2	0.2	0.004905	0.005041
11	10	2	0.2	0.005105	0.005120
13	0	2	0.2	0.004981	0.005137
9	11	2	0.2	0.004960	0.005165

Table 12: The four best NNs using the upper structure variant 2 data set and activation function 'logistic'.

Foundation, variant 1, activation function ‚tanh‘					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
7	10	1.614578	0.161457	0.010901	0.010182
10	0	1.171123	0.117112	0.017736	0.010645
9	12	0.945435	0.094543	0.017116	0.010927
8	11	1.375089	0.137508	0.015238	0.011327

Table 13: The four best NNs using the foundation variant 1 data set and activation function ‘tanh’.

Foundation, variant 1, activation function ‚logistic‘					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
3	10	2	0.2	0.008431	0.007469
14	8	1.796985	0.179698	0.009421	0.007585
14	6	1.796985	0.179698	0.009594	0.007615
11	3	1.796985	0.179698	0.009948	0.008222

Table 14: The four best NNs using the foundation variant 1 data set and activation function ‘logistic’.

Foundation, variant 2, activation function ‚tanh‘					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
7	11	0.805199	0.080519	0.014943	0.011071
13	13	0.235136	0.023513	0.012524	0.011261
7	13	0.945435	0.094543	0.014283	0.012457
8	14	0.763239	0.076323	0.014488	0.012503

Table 15: The four best NNs using the foundation variant 2 data set and activation function ‘tanh’.

Foundation, variant 2, activation function ‚logistic‘					
Layer 1	**Layer 2**	**eta_start**	**eta_end**	**E**	**Etest**
11	5	1.89577	0.189577	0.009876	0.009778
14	14	1.89577	0.189577	0.010255	0.010519
12	12	2	0.2	0.010623	0.010632
6	10	1.89577	0.189577	0.010123	0.010650

Table 16: The four best NNs using the foundation variant 2 data set and activation function ‘logistic’.

9.3 Selecting the Best Performing Neural Network configuration

From the output files produced by the program `NN_GDA.m`, the best NN configurations are selected and further improved with manually set learning parameters. This is done individually for the upper structure and foundation.

To perform this task, another program called `NN_GDA_single_layout.m` has been created. The algorithm works very similar to the one used in `NN_GDA.m`. Instead of optimizing many different NN configurations with many different η values, only one NN configuration is tested and the value of η set by hand. The optimization process is repeated

several times with the exact same parameters but newly randomized initial weights and the best resulting NN is stored in an Excel file, including the weight matrices necessary to use the NN later on.

9.3.1 Upper Structure

While the NNs using the 'tanh' activation function were probably optimized nearly to their full potential (indicated by optimal η values somewhere in the middle of the applied range), the NNs using the 'logistic' activation function can possibly still be improved by using η values higher than 2.0. In spite of this, the 'logistic' function already proved to be a much better choice and therefore the results from Table 9 and Table 11 are discarded.

Taking a closer look at Table 10, it is apparent that there might be a better NN configuration using more than 16 nodes in the first layer since the maximum number of nodes tested for each layer was 16 (twice the number of input variables, 8). However, since the results from Table 12 are consistently better than those of Table 10, it is improbable that the final NN will use variant 1 of the data sets. Therefore, Table 10 is discarded as well.

From the remaining NN configurations, the three best performing ones are chosen to be further improved. Using `NN_GDA_single_layout.m` with various learning parameters, the best results that could be achieved from the considered NN configurations are shown in Table 17 and the full output files can be found on the USB drive (see appendix 13.1).

Variant	Layer 1	Layer 2	Function	eta_start	eta_end	E	Etest
2	5	15	logistic	6	0.6	0.004414	0.003952
2	11	10	logistic	9.5	0.95	0.003798	0.003814
2	13	0	logistic	12	1.2	0.003706	0.003372

Table 17: Selected and improved NN configurations for the upper structure data set.

As can be seen in Table 17, the final NN to be used for estimating the construction material quantities of the upper structure is the NN with one hidden layer counting 13 nodes using the 'logistic' activation function, resulting in a test error of 0.003372.

9.3.2 Foundation

Like the NN configurations found for the upper structure, the NNs using the 'tanh' activation function were optimized using η values somewhere in the middle of the applied range, indicating that they probably can't be further optimized using values higher than 2.0 or lower than 0.01. Also, some of the NN configurations with the 'logistic' activation function were

optimized with `eta_start=2.0`. Furthermore, the NNs using data set variant 1 and activation function 'logistic' consistently performed better than the others. Therefore, the best three NNs from Table 14 were selected. The NNs with the lowest test error that could be found using `NN_GDA_single_layout.m` are listed in Table 18:

Variant	Layer 1	Layer 2	Function	eta_start	eta_end	E	Etest
1	3	10	logistic	10	1	0.003745	0.003369
1	14	8	logistic	15	1.5	0.002791	0.002920
1	14	6	logistic	14	1.4	0.002923	0.003355

Table 18: Selected and improved NN configurations for the foundation data set.

The final NN to be used for the foundation is the NN consisting of two hidden layers of 14 and 8 nodes each using the 'logistic' activation function, resulting in a test error of 0.002920.

9.3.3 Calculating the Output of a Given Neural Network

In order to actually use the NNs found in the previous subsections, a program is necessary to calculate the output of a new set of input values. The program is called `NN_run.m` and requires three input parameters:

- `NN_filename`: The filename of the Excel file containing the NN produced by `NN_GDA_single_layout.m`, for example `'NN_configuration.xlsx'`.
- `NN_sheet`: The sheet in the Excel file containing the NN, usually `1`.
- `I0`: A row vector containing the inputs whose outputs are to be calculated. The format and order of the elements is the same as the one found in the data sets used to optimize the NN, for example `[1,0,0,0,5,50,0.7,20]` could be used for a NN trained by variant 2 of the upper structure data set.

The outputs are given directly in the MatLab console. The calculations done by the program are shown in appendix 13.3.

10 Discussion

10.1 Impact of the Number of Nodes on the Resulting Error

A point of interest is the relationship between the chosen NN layout and the resulting training and test errors after optimization. In order to examine this relationship, the error values of the NNs generated by `NN_GDA.m` for variant 1 of the upper structure data set have been plotted against the NNs ordered by the number of nodes.

In Figure 5 and Figure 6, the horizontal axis represents the respective NNs by number of nodes in the first layer and ordered by the number of nodes in the second layer.

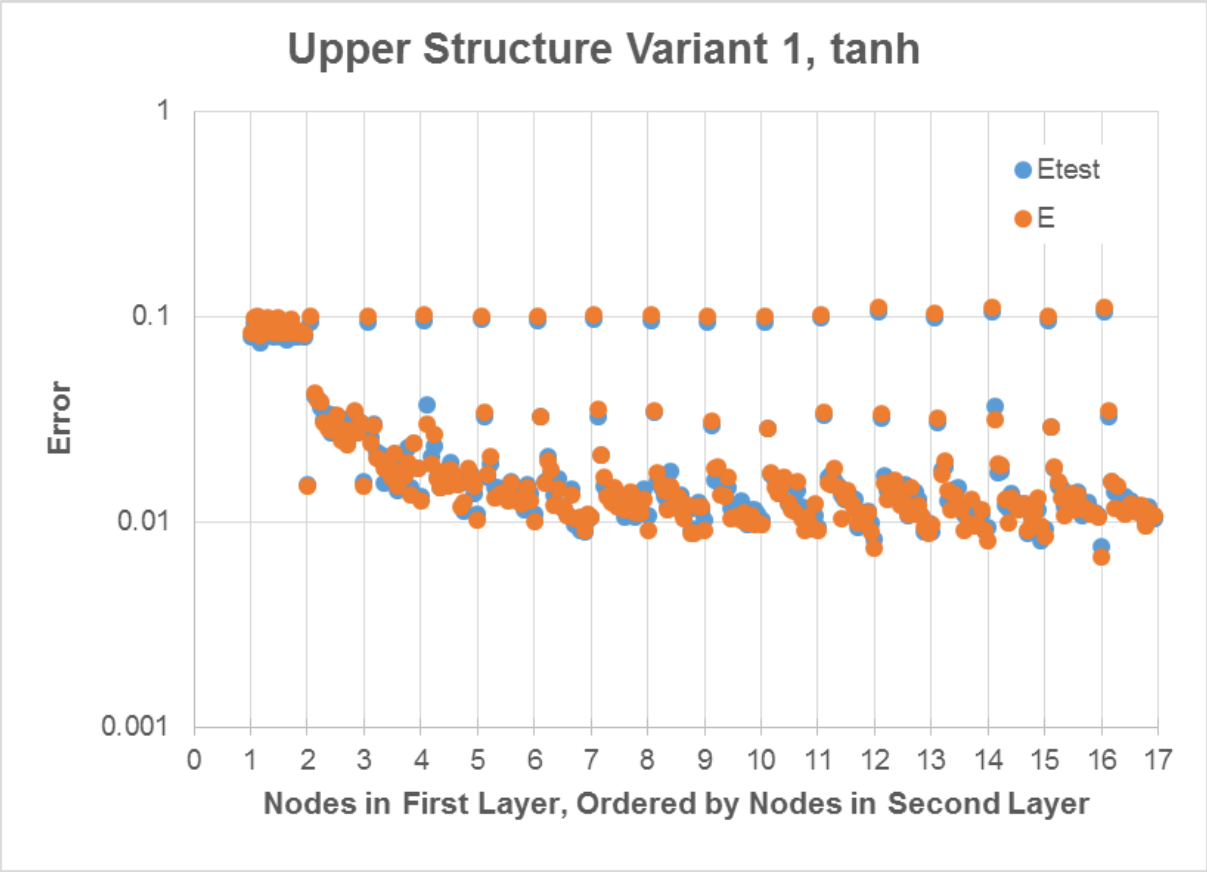

Figure 5: Errors vs. ordered number of nodes for the upper structure variant 1 data set and tanh activation function.

Figure 5 uses the results from the upper structure variant 1 data set using the hyperbolic tangent activation function. The same graph containing the results using the logistic activation function is shown in Figure 6.

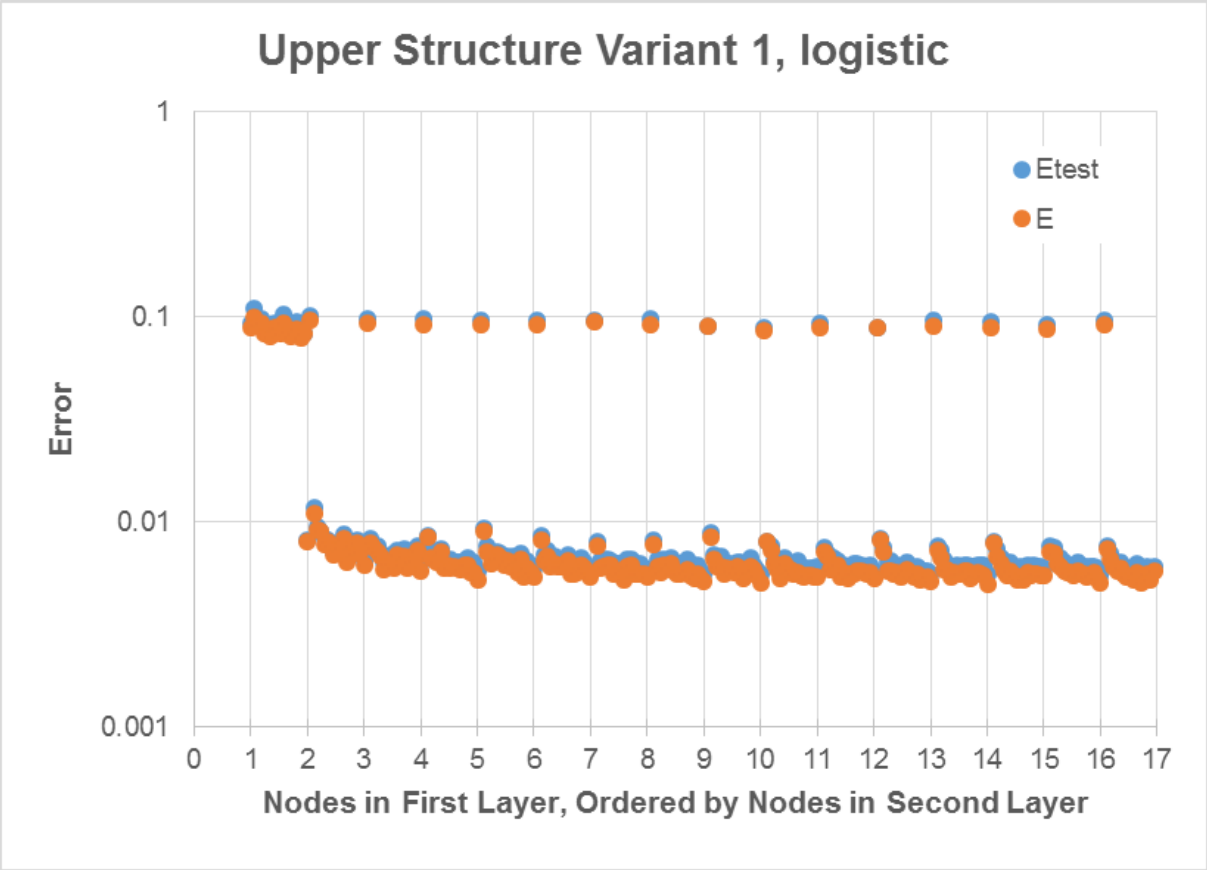

Figure 6: Errors vs. ordered number of nodes for the upper structure variant 1 data set and logistic activation function.

Both figures show a similar picture, clearly indicating that NNs with just one node in the first or second layer perform the worst, resulting in errors of about 0.1. Figure 5 also shows that the error improves quickly by including up to about five nodes in the first layer and more than two nodes in the second layer, where the NNs reach an error of nearly 0.01.

This information can also be seen by plotting the errors against the NNs ordered by the total number of nodes, which is shown in Figure 7 and Figure 8.

Graphically representing the errors like this, there are clear lines visible which can be associated with the number of nodes in the first and second layer determined in Figure 5 and Figure 6. Additionally, it can be seen that NNs with only one layer seem to perform the best when using the hyperbolic tangent activation function. In general, the effects of the number of nodes on the resulting errors are much more pronounced when using the hyperbolic tangent activation function, showing a broad range of errors depending on the chosen NN layout.

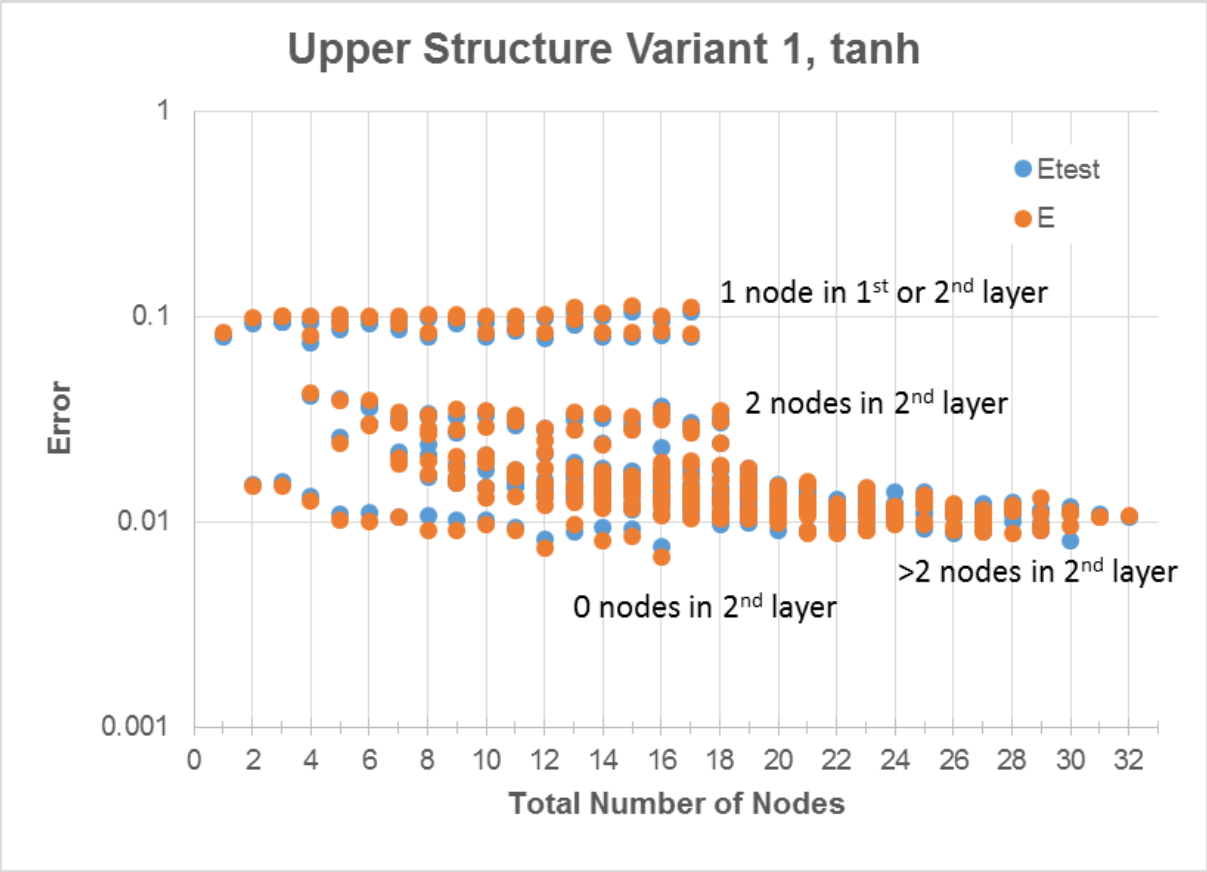

Figure 7: Errors vs. total number of nodes for the upper structure variant 1 data set and tanh activation function.

The effects of overtraining, i.e. a decreasing training error coupled with an increasing test error when adding more nodes to the NN, are not visible in these diagrams. The reason for this is suspected to be the fact that the NN layouts examined had a heuristically chosen upper limit for the number of nodes in each layer equalling two times the number of input variables. Since this number was determined based on what was found in the literature [Hegazy 1994], it would only make sense for such NNs not to suffer from overtraining.

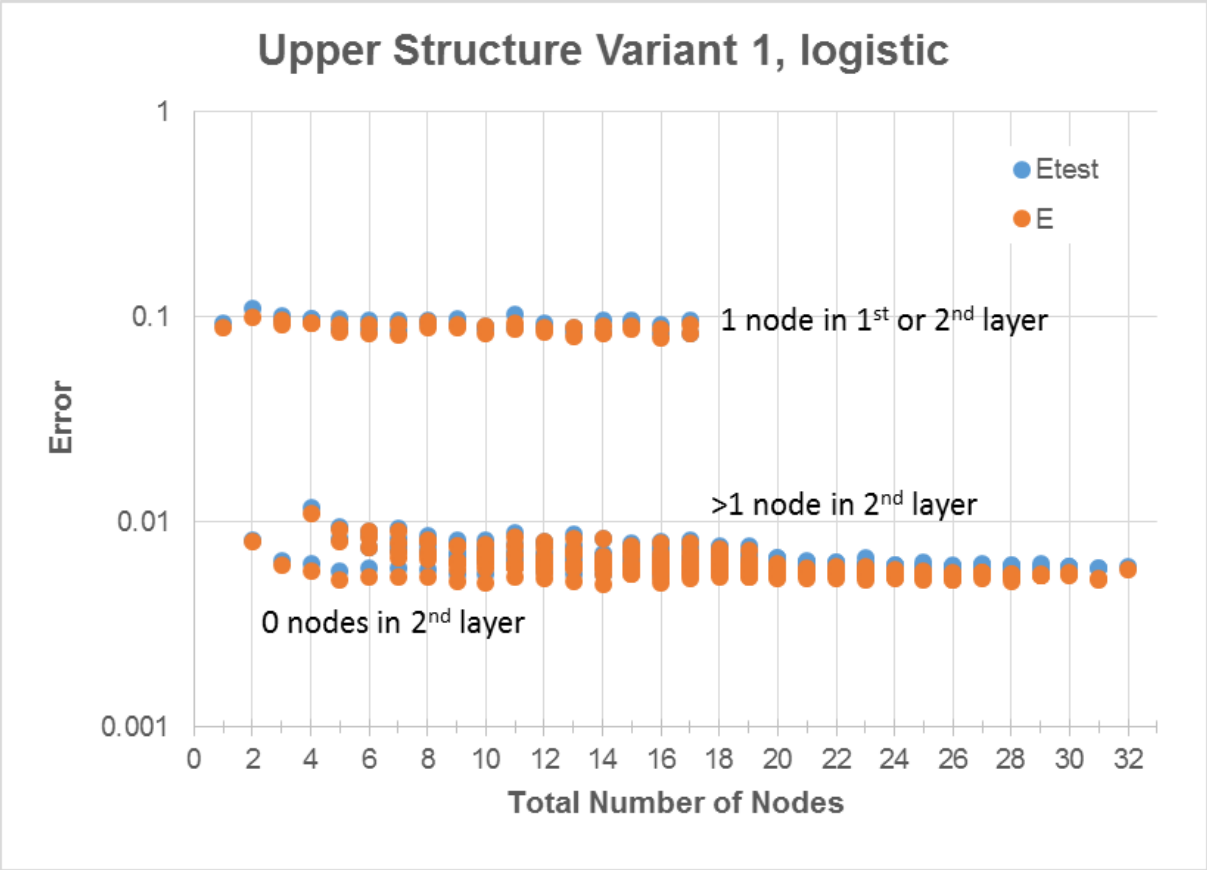

Figure 8: Errors vs. total number of nodes for the upper structure variant 1 data set and logistic activation function.

10.2 Input Variable Importance

The input variable importance is calculated according to the connection weight approach specified by Olden [Olden 2004]. This means that, starting at the output buffer with a value of 1, 'importance signals' are sent back along the connections. The signals are multiplied by the weights attached to the connections and all signals reaching a node are summed to determine the signal sent onwards from that node. Finally, the summed signals at the input buffer represent the importance of the corresponding input variable. Mathematically, this can be expressed by the following recursive formulas, starting at the vector $\boldsymbol{S_h}$ containing the importance signals of the output buffer (layer h) and then calculating the signals $\boldsymbol{S_g}$ of all preceding layers g:

$$\boldsymbol{S_h} = \boldsymbol{1_h}$$

$$\boldsymbol{S_g} = \boldsymbol{W_{g+1}} \cdot \boldsymbol{S_{g+1}}$$

In these formulas, the vector $\mathbf{1}_h$ is a column vectors of elements 1 according to the number of outputs of the NN and the last row of each weight matrix $\boldsymbol{W}$ is removed (since bias nodes can't back-propagate importance signals). Finally, the vector $\boldsymbol{S_0}$ contains the importance values of the input variables.

It is essential to note that, using these formulas, the importance of the input variables are related to all outputs of the NN weighted equally. With slight modifications of the formulas it is possible to calculate the input variable importance with respect to a single output variable or any arbitrary set of weighted output variables. In this work, we limit ourselves to the importance values as determined by the calculations above.

The connection weight approach has been implemented in `NN_connection_weights.m` and then applied to the best NN for the upper structure and foundation. The importance values are sorted by absolute value to create a ranking of the most important input variables. The results of the final upper structure NN containing one layer of 13 nodes are shown in Table 19:

Rank	Input variable	Importance
1	H2	103.1
2	C1	-92.3
3	Stages	67.2
4	C2	-58.9
5	H1	57.9
6	Wind speed	19.8
7	Soil bearing capacity	4.5
8	Ground acceleration	-3.7

Table 19: Input variable importance of the final NN for the upper structure.

It can be seen that the choice of construction material and number of strings and stages has the largest impact on the resulting outputs. The site parameters, however, could possibly be neglected in the creation of a new NN. The final foundation NN containing 14 and 8 nodes in two layers shows similar results, see Table 20:

Rank	Input variable	Importance
1	Total height	774.0
2	Soil bearing capacity	-604.0
3	Single string	-433.6
4	Double string	367.1
5	Ground height	-205.6
6	Wind speed	12.8
7	Ground acceleration	2.3

Table 20: Input variable importance of the final NN for the foundation.

The main difference is that the soil bearing capacity is much more important for the material quantities of the foundation and that there is an additional important parameter *ground height*.

10.3 Comparison to Case-Based Reasoning

In order to compare the developed NN to other methods of estimation, a case-based reasoning (CBR) model was created. The CBR model implemented is similar to the one used in [Kim 2004] and is outlined in the following paragraph.

The implemented CBR model is a method based on experience and memory, consisting of the following processes [Kim 2004]:

1. Old problem *cases* (data points) are stored in a *case base* (data set).
2. New cases are compared to cases in the case base and those with the highest *similarity score* are retrieved.
3. The outputs of the new case are estimated based on the retrieved cases.

In this application, the case base is variant 2 of the upper structure data set, which was used to create the best performing upper structure NN (see chapter 9.3.1). In order to calculate the similarity score between two cases, the following formula was used:

$$S = \frac{\sum_i f_i \cdot w_i}{\sum_i w_i}$$

Where f_i is the output of the similarity function $f(N_i,O_i)$, comparing attribute i of new case N to a case in the case base O: For the binary input variables (see chapter 8.2.2), f_i is 1 if the values of the new and old case are the same and 0 otherwise. For numerical input variables, f_i is 1 if the deviation is less than or equal to a tolerance of 10%:

$$\frac{N_i - O_i}{O_i} \leq 0.1$$

If the deviation is more than 10%, f_i is 0. The similarity function of an input variable is weighted by w_i to properly represent the importance of certain input variables. When the similarity score S has been calculated between the new case and each of the old cases in the case base, the outputs of the new case are chosen identical to those of the case with the highest similarity score. If there are several cases with the highest similarity score, their outputs are averaged. This procedure is implemented in the program `CBR.m` (see appendices 13.1 and 13.2.6).

In order to compare the developed NN model to the CBR model, 100 cases of each of the building types of the upper structure variant 2 data set were removed from the case base and presented as new cases. Then, the error of the CBR model was calculated using the same error function as the one used for the NN:

$$E = \frac{1}{n}\sum_{n}\sum_{j}\left(\frac{O_{nj} - O_{nj,true}}{O_{j,true,max} - O_{j,true,min}}\right)^2$$

The error was calculated using the program `CBR_error.m` (see appendices 13.1 and 13.2.7). With a tolerance of 0.1 and all input variables weights equal to 1, the program calculated an error of 0.005299. Using the absolute values of the input variable importance of Table 19 as weights, the resulting error was 0.004151.

Comparing this to the error of the developed NN of 0.003372, it can be said that the NN performs better than the CBR model. Using the input variables importance as weights for the CBR model lead to improvements, but still resulted in a higher error than the NN. Concluding this, the NN model performs better in regards to accuracy but requires more effort to implement properly.

11 Conclusions and Outlook

During the step *selecting the best performing NN configurations*, when NNs were generated using manually entered learning parameters, it became apparent that there is a lot of chance involved in generating random weight matrices that lead to good optimization. In addition, the graphs shown in chapter 10.1 clearly indicate which number of nodes in a NN led to a low error. With this knowledge, the proposed model could be changed to considerably shorten the whole process of creating a practical NN, for example by lowering the number of iterations used in *generating possible NN configuration* and then determining which NNs to further improve graphically.

In general, the results of the developed NN model are satisfactory, especially when compared to the employed CBR model which is a tried-and-true method used for estimation. The NN approach has proved to be a useful tool for applications in the field of civil engineering.

12 References

[Bishop 1994] Bishop, C. M. (1994). Neural networks and their applications. *Review of scientific instruments*, *65*(6), 1803-1832.

[Boussabaine 1996] Boussabaine, A. H. (1996). The use of artificial neural networks in construction management: a review. *Construction Management & Economics*, *14*(5), 427-436.

[Hegazy 1994] Hegazy, T., Fazio, P., & Moselhi, O. (1994). Developing Practical Neural Network Applications Using Back-Propagation. *Computer-Aided Civil and Infrastructure Engineering*, *9*(2), 145-159.

[Hegazy 1998] Hegazy, T., & Ayed, A. (1998). Neural network model for parametric cost estimation of highway projects. *Journal of Construction Engineering and Management*, *124*(3), 210-218.

[Kim 2004] Kim, G. H., An, S. H., & Kang, K. I. (2004). Comparison of construction cost estimating models based on regression analysis, neural networks, and case-based reasoning. *Building and environment*, *39*(10), 1235-1242.

[Olden 2002] Olden, J. D., & Jackson, D. A. (2002). Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, *154*(1), 135-150.

[Olden 2004] Olden, J. D., Joy, M. K., & Death, R. G. (2004). An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, *178*(3), 389-397.

[Melanie 1999] Melanie, M. (1999). An introduction to genetic algorithms. *Cambridge, Massachusetts London, England, Fifth printing*, *3*.

13 Appendix

13.1 Electronic Documents

The accompanying USB drive contains the electronic documents of this project work. This includes an electronic versions of this report, the presentation, figures, MatLab files and Excel files.

The folders in alphabetical order on the USB drive contain the following files:

`\CODE` – All the input files used to run the programs, the programs themselves and the produced output files. Explained in detail in appendix 13.2.

`\FIGURES\JPEG` – All figures of this report in jpeg format. The numbers in the beginning of the filename indicate the number of the figure in this report.

`\FIGURES\SOURCE` – All figures of this report in the source format. The numbers in the beginning of the filename indicate the number of the figure in this report.

`\POSTER` – The poster in pdf and pptx format.

`\PRESENTATION` – The presentation in pdf and pptx format.

`\PROJECT_SCHEDULE` – The final project schedule in pdf and docx format.

`\REPORT` – This report in pdf and docx format.

`\TABLES\JPEG` – All tables in this report in jpeg format. The numbers in the beginning of the filename indicate the number of the table in this report.

`\TABLES\SOURCE` – Empty. All tables were created directly in the docx file of the report, therefore there are no source files for the tables.

`\WORKFILES` – The original data set provided by the supervisors and a file containing the calculations for the coefficients of correlation done in Excel.

13.2 Program Documentation

13.2.1 Overview

The general workflow intended when using the programs coded in MatLab R2012a is shown in Figure 9:

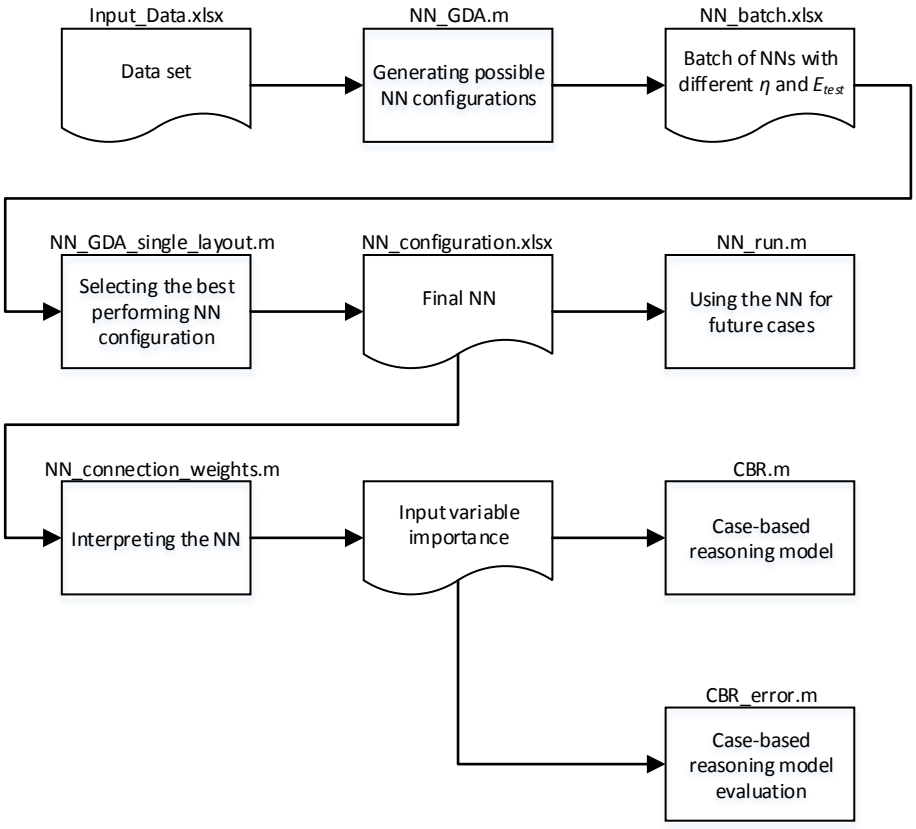

Figure 9: Documents, programs and workflow of the implementation.

The data set is an Excel file containing the inputs and outputs of the training and test data. Using this data, `NN_GDA.m` performs the generalized delta rule on a multitude of possible NN configurations to estimate their performance and optimal learning parameters. This information is stored in an Excel file and will be used to select a specific NN configuration for further optimization. Once a NN configuration has been selected, `NN_GDA_single_layout.m` is used to produce an optimized NN, stored in an Excel file. Finally, `NN_run.m` can be used on the produced NN to calculate the outputs to a new set of inputs. To analyse the NN, `NN_connection_weights.m` can be run to calculate the input variable importance according to the connection weights approach. These importance values can be used in `CBR.m`, a program using the case-based reasoning approach to solve the same problem as the NN. `CBR_error.m` calculates the error of the case-based reasoning model with specific weights.

All other MatLab files included in this project are external functions used to make the program code more comprehensible. They are called by the programs listed above and don't

require any user interaction. All MatLab files and called Excel files should stay in the same directory and should not be renamed for the programs to run properly.

The Excel files which acted as inputs and outputs in this project work are the following, in alphabetical order:

`Data_foundation_var1.xlsx` – Variant 1 of the data sets of the foundation.

`Data_foundation_var2.xlsx` – Variant 2 of the data sets of the foundation.

`Data_upperstruct_var1.xlsx` – Variant 1 of the data sets of the upper structure.

`Data_upperstruct_var2.xlsx` – Variant 2 of the data sets of the upper structure.

`NN_batch_foundation_var1_logistic.xlsx` – Output of `NN_GDA.m` using variant 1 of the data sets of the foundation and logistic activation function.

`NN_batch_foundation_var1_tanh.xlsx` – Output of `NN_GDA.m` using variant 1 of the data sets of the foundation and hyperbolic tangent activation function.

`NN_batch_foundation_var2_logistic.xlsx` – Output of `NN_GDA.m` using variant 2 of the data sets of the foundation and logistic activation function.

`NN_batch_foundation_var2_tanh.xlsx` – Output of `NN_GDA.m` using variant 2 of the data sets of the foundation and hyperbolic tangent activation function.

`NN_batch_upperstruct_var1_logistic.xlsx` – Output of `NN_GDA.m` using variant 1 of the data sets of the upper structure and logistic activation function.

`NN_batch_upperstruct_var1_tanh.xlsx` – Output of `NN_GDA.m` using variant 1 of the data sets of the upper structure and hyperbolic tangent activation function.

`NN_batch_upperstruct_var2_logistic.xlsx` – Output of `NN_GDA.m` using variant 2 of the data sets of the upper structure and logistic activation function.

`NN_batch_upperstruct_var2_tanh.xlsx` – Output of `NN_GDA.m` using variant 1 of the data sets of the upper structure and hyperbolic tangent activation function.

`NN_config_3_10.xlsx` – Output of `NN_GDA_single_layout.m` corresponding to the NN with 3 and 10 nodes in the first and second layer corresponding to Table 18.

`NN_config_5_15.xlsx` – Output of `NN_GDA_single_layout.m` corresponding to the NN with 5 and 15 nodes in the first and second layer corresponding to Table 17.

`NN_config_11_10.xlsx` – Output of `NN_GDA_single_layout.m` corresponding to the NN with 11 and 10 nodes in the first and second layer corresponding to Table 17.

`NN_config_13_0.xlsx` – Output of `NN_GDA_single_layout.m` corresponding to the NN with 13 and 0 nodes in the first and second layer corresponding to Table 17.

`NN_config_14_6.xlsx` – Output of `NN_GDA_single_layout.m` corresponding to the NN with 3 and 10 nodes in the first and second layer corresponding to Table 18.

`NN_config_14_8.xlsx` – Output of `NN_GDA_single_layout.m` corresponding to the NN with 3 and 10 nodes in the first and second layer corresponding to Table 18.

13.2.2 NN_GDA.m

The program `NN_GDA.m` requires the following user inputs in order to run:

- `input_filename`: The filename of the Excel file containing the data set, for example `'Input_Data.xlsx'`.
- `data_set`: The name of the data set, either `'upperstruct'` or `'foundation'`.
- `input_sheet`: The sheet in the Excel file containing the data set, usually `1`.
- `input_range_string`: The cell range in the Excel file containing the data set, for example `'A2:L4507'`.
- `input_range`: The range of columns containing the input variables, for example `[1:8]` for columns 1 to 8.
- `input_range_type`: The range of columns containing binary data (the building types), for example `[1:4]` for columns 1 to 4.
- `output_range`: The range of columns containing the output variables, for example `[9:12]` for columns 9 to 12.
- `output_filename`: The output of the program will be written to a new Excel file with this name, for example `'NN_batch.xlsx'`.
- `output_sheet`: The sheet in the output Excel file, usually `1`.
- `output_comment`: A comment to be included in the output Excel file in string format.
- `activation_function`: The activation function to be used for all nodes, either `'tanh'` or `'logistic'`.

- `n`: The number of data points per building type to use for training, for example `100` will result in a total of 400 data points for the upper structure since there are four building types.
- `ntest`: The number of data points per building type to use for calculating the performance of an optimized NN configuration.
- `p_eta_start`: The upper limit to use for learning rate η in the first iteration of the generalized delta rule, for example `2`.
- `p_eta_end`: The lower limit to use for learning rate η in the first iteration of the generalized delta rule, for example `0.01`.
- `eta_ratio`: The ratio of learning rate η of the first iteration to that of the last iteration of the generalized delta rule, for example `10`.
- `mu`: The learning rate μ to use for the generalized delta rule, for example `0.9`.
- `NN_iterations`: The number of iterations where a given NN configuration is optimized (middle loop *2-3-4-6* in Figure 4), for example `100`.
- `GDA_iterations`: The number of iterations of the generalized delta rule (inner loop *3-4-5* in Figure 4), for example `400`.
- `max_nodes`: The maximum number of nodes in the two layers, for example `[5,8]` to generate $5 \cdot (8 + 1) = 45$ different NN configurations with a maximum of 5 nodes in the first layer and 8 nodes in the second layer. An empty matrix `[]` will set it to the default of two times the number of input variables for each layer.

The output is stored in the file named by `output_filename`, containing the NN configurations optimized as well as the best η and errors that could be found. In addition, all input values are stored so the results can be reproduced.

13.2.3 NN_GDA_single_layout.m

The program `NN_GDA.m` requires the following user inputs in order to run:

- `input_filename`: The filename of the Excel file containing the data set, for example `'Input_Data.xlsx'`.
- `data_set`: The name of the data set, either `'upperstruct'` or `'foundation'`.
- `input_sheet`: The sheet in the Excel file containing the data set, usually `1`.
- `input_range_string`: The cell range in the Excel file containing the data set, for example `'A2:L4507'`.

- `input_range`: The range of columns containing the input variables, for example `[1:8]` for columns 1 to 8.
- `input_range_type`: The range of columns containing binary data (the building types), for example `[1:4]` for columns 1 to 4.
- `output_range`: The range of columns containing the output variables, for example `[9:12]` for columns 9 to 12.
- `output_filename`: The output of the program will be written to a new Excel file with this name, for example `'NN_batch.xlsx'`.
- `output_sheet`: The sheet in the output Excel file, usually `1`.
- `output_comment`: A comment to be included in the output Excel file in string format.
- `activation_function`: The activation function to be used for all nodes, either `'tanh'` or `'logistic'`.
- `n`: The number of data points per building type to use for training, for example `100` will result in a total of 400 data points for the upper structure since there are four building types.
- `ntest`: The number of data points per building type to use for calculating the performance of an optimized NN configuration.
- `eta_start`: The learning rate η in the first iteration of the generalized delta rule, for example `2`.
- `eta_end`: The learning rate η in the last iteration of the generalized delta rule, for example `0.2` or `eta_start/10`.
- `mu`: The learning rate μ to use for the generalized delta rule, for example `0.9`.
- `NN_iterations`: The number of times that the weights will be reinitialized with random values, for example `100` if the best NN of 100 separate optimizations of the same NN configuration has to be found.
- `GDA_iterations`: The number of iterations of the generalized delta rule (inner loop *3-4-5* in Figure 4), for example `400`.
- `Hidden_Layers`: The number of nodes in the hidden layers, for example `[5,8]` to optimize a NN configuration with 5 nodes in the first layer and 8 nodes in the second layer. Requires at least one element and must not contain any zeroes (`0`) and must not be empty. More than two hidden layers are possible.

The output is stored in the file named by `output_filename`, containing the weights of the best optimized NN that could be found. In addition, all input values are stored so the results can be reproduced.

13.2.4 NN_run.m

The program `NN_run.m` requires the following user inputs in order to run:

- `NN_filename`: The filename of the Excel file containing the NN produced by `NN_GDA_single_layout.m`, for example `'NN_configuration.xlsx'`.
- `NN_sheet`: The sheet in the Excel file containing the NN, usually `1`.
- `I0`: A row vector containing the inputs whose outputs are to be calculated. The format and order of the elements is the same as the one found in the data sets used to optimize the NN, for example `[1,0,0,0,5,50,0.7,20]` could be used for a NN trained by variant 2 of the upper structure data set.

The outputs of the NN corresponding to the entered inputs are displayed directly in the MatLab console.

13.2.5 NN_connection_weights.m

The program `NN_connection_weights.m` requires the following user inputs in order to run:

- `NN_filename`: The filename of the Excel file containing the NN produced by `NN_GDA_single_layout.m`, for example `'NN_configuration.xlsx'`.
- `NN_sheet`: The sheet in the Excel file containing the NN, usually `1`.

The outputs are displayed directly in the MatLab console, containing the importance and importance rank of each input variable.

13.2.6 CBR.m

The program `CBR.m` requires the following user inputs in order to run:

- `input_filename`: The filename of the Excel file containing the case base, for example `'Input_Data.xlsx'`.
- `input_sheet`: The sheet in the Excel file containing the case base, usually `1`.
- `input_range_string`: The cell range in the Excel file containing the case base, for example `'A2:L4507'`.

- `input_range_ooal`: The range of columns containing binary data (the building types), for example `[1:4]` for columns 1 to 4.
- `input_range_num`: The range of columns containing numerical data (all other input variables), for example `[5:8]` for columns 5 to 8.
- `output_range`: The range of columns containing the output variables, for example `[9:12]` for columns 9 to 12.
- `I`: A row vector containing a new case whose outputs are to be calculated. The format and order of the elements is the same as the one found in the case, for example `[1,0,0,0,5,50,0.7,20]` could be used for a case-based reasoning model applied to variant 2 of the upper structure data set.
- `W`: A row vector containing the weights attached to the input variables. The order of the elements is the same as the one entered in `I`, for example `[1,1,1,1,1,1,1,1]` could be used for a case-based reasoning model applied to variant 2 of the upper structure data set where each input variable is weighted equally.
- `tolerance`: The matching tolerance for two cases' numerical values to be considered similar, for example `0.1` for a tolerance of 10%.

The outputs of the case-based reasoning model corresponding to the entered inputs are displayed directly in the MatLab console.

13.2.7 CBR_error.m

The program `CBR_error.m` requires the following user inputs in order to run:

- `input_filename`: The filename of the Excel file containing the case base, for example `'Input_Data.xlsx'`.
- `input_sheet`: The sheet in the Excel file containing the case base, usually `1`.
- `input_range_string`: The cell range in the Excel file containing the case base, for example `'A2:L4507'`.
- `input_range_ooal`: The range of columns containing binary data (the building types), for example `[1:4]` for columns 1 to 4.
- `input_range_num`: The range of columns containing numerical data (all other input variables), for example `[5:8]` for columns 5 to 8.

- `output_range`: The range of columns containing the output variables, for example `[9:12]` for columns 9 to 12.
- `ntest`: The number of cases per building type to use for calculating the performance of the case-based reasoning model. The number shouldn't be too high since these cases will be removed from the case base in order to calculate the resulting error.
- `W`: A row vector containing the weights attached to the input variables. The order of the elements is the same as the one entered in `I`, for example `[1,1,1,1,1,1,1,1]`could be used for a case-based reasoning model applied to variant 2 of the upper structure data set where each input variable is weighted equally.
- `tolerance`: The matching tolerance for two cases' numerical values to be considered similar, for example `0.1` for a tolerance of 10%.

The resulting error (using the same error function as the NN model) of the chosen case-based reasoning model is displayed directly in the MatLab console.

13.3 Calculating the Neural Network Output

In this section, the exact calculations necessary to map a given set of inputs to the outputs of a NN are presented. Bold symbol denote matrices or vectors.

I is the input matrix containing the input data with *n* rows and *m* columns. *n* is the number of records from the data, *m* the number of input variables.

$$\boldsymbol{I_0} = \begin{pmatrix} I_{0,11} & I_{0,12} & \dots & I_{0,1m} \\ I_{0,21} & I_{0,22} & \dots & I_{0,2m} \\ \dots & \dots & \dots & \dots \\ I_{0,n1} & I_{0,n2} & \dots & I_{0,nm} \end{pmatrix}$$

The elements of the input buffer matrix $\boldsymbol{X_0}$ are calculated using the minimum and maximum values of the j^{th} column of $\boldsymbol{I_0}$, effectively scaling the inputs to a range of [-1,1].

$$X_{0,ij} = \frac{2\left(I_{0,ij} - I_{0,min,j}\right)}{I_{0,max,j} - I_{0,min,j}} - 1$$

The matrix $\boldsymbol{X_0}$ will show *n* rows for *n* records and $l=m+1$ columns for *m* input variables plus a bias of value 1.

$$\boldsymbol{X_0} = \begin{pmatrix} X_{0,11} & X_{0,12} & \dots & X_{0,1m} & 1 \\ X_{0,21} & X_{0,22} & \dots & X_{0,2m} & 1 \\ \dots & \dots & \dots & \dots & \dots \\ X_{0,n1} & X_{0,n2} & \dots & X_{0,nm} & 1 \end{pmatrix}$$

The input buffer is connected to the first hidden layer using the weights $W_{1,ij}$ connecting input variable *i* to node *j* of the first hidden layer. The weight matrix $\boldsymbol{W_1}$ has *l* rows and *a* columns for a total of *a* nodes in the first hidden layer.

$$\boldsymbol{W_1} = \begin{pmatrix} W_{1,11} & W_{1,12} & \dots & W_{1,1a} \\ W_{1,21} & W_{1,22} & \dots & W_{1,2a} \\ \dots & \dots & \dots & \dots \\ W_{1,l1} & W_{1,l2} & \dots & W_{1,la} \end{pmatrix}$$

$\boldsymbol{I_1}$ contains the sum of signals received by each node of the first hidden layer for each data record and has *n* rows and *a* columns resulting from matrix multiplication.

$$\boldsymbol{I_1} = \boldsymbol{X_0} \cdot \boldsymbol{W_1}$$

The incoming signals are then processed using the activation function *f(x)* (hyperbolic tangent or logistic) to produce the outputs of the first hidden layer, stored in matrix $\boldsymbol{X_1}$.

$$X_{1,ij} = f(I_{1,ij})$$

The matrix $\boldsymbol{X_1}$ will show *n* rows for *n* records and *b=a+1* columns for *a* nodes in the first hidden layer plus a bias of value 1.

$$\boldsymbol{X_1} = \begin{pmatrix} X_{1,11} & X_{1,12} & \dots & X_{1,1a} & 1 \\ X_{1,21} & X_{1,22} & \dots & X_{1,2a} & 1 \\ \dots & \dots & \dots & \dots & \dots \\ X_{1,n1} & X_{1,n2} & \dots & X_{1,na} & 1 \end{pmatrix}$$

This is repeated for the second hidden layer using weights $\boldsymbol{W_2}$ to calculate the summed signals $\boldsymbol{I_2}$ which are processed by the nodes to produce the outputs $\boldsymbol{X_2}$.

The outputs of the last layer *h*, stored in output buffer matrix $\boldsymbol{X_h}$ (without bias) are scaled to produce the output matrix $\boldsymbol{O}$ which has *n* rows and columns matching the number of outputs to be produced by the NN. To calculate the output O_{ij}, the minimum and maximum values of possible outputs are required.

$$O_{ij} = \frac{(X_{h,ij} - a_f)(O_{max,j} - O_{min,j})}{(b_f - a_f)} + O_{min,j}$$

This is the reverse of the calculations used to scale in the input data, adjusted to the output range $[a_f,b_f]$ of the chosen activation function.

13.4 Calculating the Weight Updates

In this section, the exact calculations of the generalized delta rule (error back-propagation) are presented. Bold symbol denote matrices or vectors.

E is the sum-of-squares error which can be used to evaluate the performance of a NN and the goal of the weight updates is to minimize this error. To calculate this error, the differences between actual and desired output of the output buffer are squared and summed. Using the same notation as in the preceding subsection, the error function reads:

$$E = \frac{1}{2}\sum_{n}\sum_{j}\left(X_{h,nj} - X_{h,nj,true}\right)^2$$

The gradient descent method uses the gradient of *E* which is calculated using the back-propagated error signals *D*. For the weights connected to the output buffer, $\boldsymbol{W_h}$, the calculations are:

$$D_{h,nj} = f'(X_{h,nj}) \cdot \left(X_{h,nj} - X_{h,nj,true}\right)$$

$$\nabla E_{h,ij} = \sum_{n} D_{h,nj} \cdot X_{h-1,ni}$$

Where *f'(x)* is the derivative of the chosen activation function *f(x)*. The calculations of the error signals differ for the layer before the last layer.

$$D_{h-1,nj} = f'(X_{h,nj}) \cdot ((\boldsymbol{W_h} \cdot \boldsymbol{D_h^\top})^\top)_{nj}$$

$$\nabla E_{h-1,ij} = \sum_{n} D_{h-1,nj} \cdot X_{h-2,ni}$$

The gradient of all other layers can be calculated using the following formulas after the last column of $\boldsymbol{D}$ has been removed (since error signals from the biases aren't back-propagated):

$$D_{g,nj} = f'(X_{h,nj}) \cdot \left(\left(\boldsymbol{W_{g+1}} \cdot \boldsymbol{D_{g+1}^\top}\right)^\top\right)_{nj}$$

$$\nabla E_{g,ij} = \sum_{n} D_{g,nj} \cdot X_{g-1,ni}$$

The weight matrices are then adjusted by $\boldsymbol{\Delta W}$ after each iteration. $\boldsymbol{\Delta W^{(i)}}$ denotes the weight changes of the i^{th} iteration, starting at zero:

$$\boldsymbol{\Delta W^{(0)} = 0}$$

$$\boldsymbol{\Delta W^{(i)} = -\eta \cdot \nabla E + \mu \cdot \Delta W^{(i-1)}}$$

$$\boldsymbol{W^{(i)} = W^{(i-1)} + \Delta W^{(i)}}$$

η is the learning rate and μ is the learning momentum, used to control the speed of the algorithm. An iteration ends when the weights have been updated.

13.5 Derivation of the Error Function

Due to the fact that different activation functions result in different ranges of the output buffer outputs $\boldsymbol{X_h}$, the error function $\hat{E}$ proposed in chapter 8.3 isn't suited to compare the performance of NNs with different activation functions. Since the final outputs $\boldsymbol{O}$ and related errors are comparable for all NNs with the same output variables (regardless of activation function), the goal is to express the squared error $\hat{E}_{nj}$ of a single output *j* of a single data point *n*, which is given by

$$\hat{E}_{nj} = \left(X_{h,nj} - X_{h,nj,true}\right)^2$$

as a function of the outputs $\boldsymbol{O}$. Then, the terms dependent on the chosen activation function can be eliminated to create the desired error function.

The scaling of the outputs of the output buffer $\boldsymbol{X_h}$ to $\boldsymbol{O}$ is done using the following formula (taken from appendix 13.3):

$$O_{nj} = \frac{\left(X_{h,nj} - a\right)\left(O_{max,j} - O_{min,j}\right)}{(b-a)} + O_{min,j}$$

$O_{max,j}$ and $O_{min,j}$ denote the maximum and minimum values of output variable *j* and *a* and *b* are the upper and lower limits of the range *[a,b]* of the chosen activation function. Solving this equation for $X_{h,nj}$ results in:

$$X_{h,nj} = \frac{O_{nj}(b-a) - O_{min,j}}{O_{max,j} - O_{min,j}} + a$$

This relation can be used to substitute $X_{h,nj}$ and $X_{h,nj,true}$ in $\hat{E}_{nj}$ which yields:

$$\hat{E}_{nj} = \left(\frac{O_{nj}(b-a) - O_{min,j}}{O_{max,j} - O_{min,j}} + a - \left(\frac{O_{nj,true}(b-a) - O_{min,j}}{O_{max,j} - O_{min,j}} + a\right)\right)^2$$

Simplifying this results in the following term:

$$\hat{E}_{nj} = (b-a)^2 \left(\frac{O_{nj} - O_{nj,true}}{O_{j,true,max} - O_{j,true,min}} \right)^2$$

It is apparent that only the constant factor $(b-a)^2$ is dependent on the chosen activation function. Therefore, dividing the original error function $\hat{E}$ by $n(b-a)^2/2$ yields a new error function which is normalized in regard to the activation function chosen as well as the number of samples *n* used:

$$E = \frac{2\hat{E}}{n(b-a)^2} = \frac{1}{n} \sum_n \sum_j \left(\frac{O_{nj} - O_{nj,true}}{O_{j,true,max} - O_{j,true,min}} \right)^2$$

Since this error function only relies on the outputs $\boldsymbol{O}$, it can be used to compare all NN configurations used in this work as well as the case-based reasoning model, as long as the characteristics of the output variables are the same (number, maxima and minima).

Compatibility with the chosen learning algorithm is also given since the relationship between E and $\hat{E}$ is monotonic. Minimizing one error function will also minimize the other, therefore the generalized delta rule is still appropriate for optimization of E.