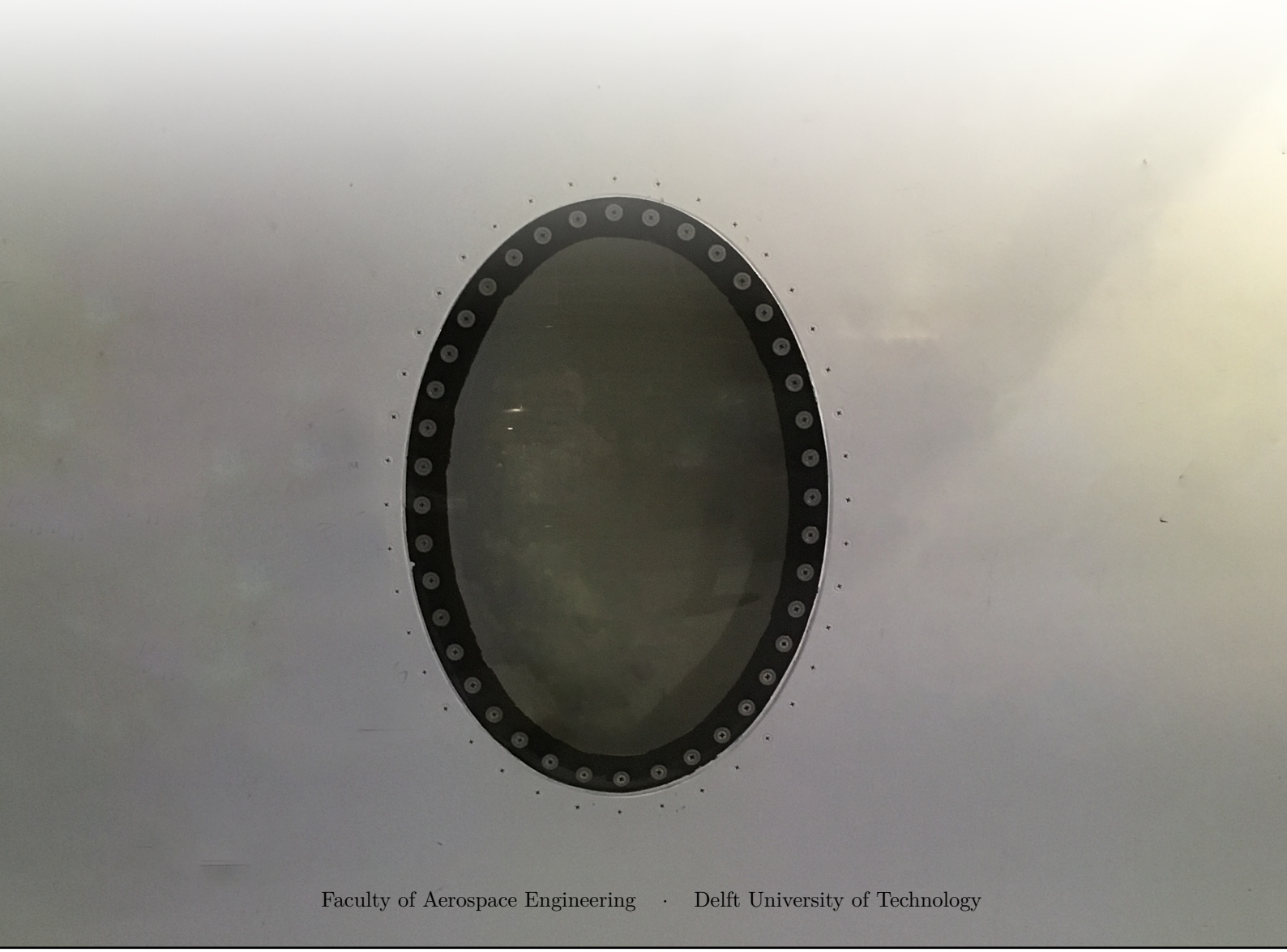
Master of Science Thesis

An Integrated Machine Learning and Finite Element Analysis Framework, applied to Composite Substructures including Damage

T.H.E. Gulikers

20-12-2018



Faculty of Aerospace Engineering · Delft University of Technology

TU Delft Delft University of Technology

An Integrated Machine Learning and Finite Element Analysis Framework, applied to Composite Substructures including Damage

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

T.H.E. Gulikers

20-12-2018

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology
Faculty of Aerospace Engineering
Department of Aerospace Structures and Materials

GRADUATION COMMITTEE

Dated: 20-12-2018

Chair holder:

Dr. ir. R. de Breuker

Committee members:

Dr. ir. B. Chen

Dr. ir. D. Zarouchas

Foreword

The master's thesis that lies in front of you is an attempt to explore a potential synergy between the fields of machine learning and structural analysis. I gratefully acknowledge and recognise the initiative of Dr. Boyang Chen in identifying the potential in this area. Ultimately, I am satisfied with the achieved results and especially, the meaningful knowledge I was able to take away from it. In these times where 'machine learning' and 'big data' are mere buzzwords to most, I had the opportunity to learn the applications and methods behind these terms.

I wish to express my gratitude towards Dr. Boyang Chen for his confidence and guidance, both of which he generously granted throughout the project. Especially during the first few months, it was a struggle to find the most promising bridge between the vast fields of machine learning and structural analysis. My sincere thanks, Boyang, for always being available for counsel, support and your occasional lectures. I honestly could not have hoped for a better supervisor and mentor.

Special thanks go to my friends in Delft for the enjoyable evenings and great, endless discussions: I'm quite sure we solved a world-problem or two during the numerous 'Thursday drinks'. Furthermore, I am grateful towards my partner Katleen for sharing the good and bad times which a thesis inevitably brings. Finally and most importantly, I am deeply grateful to my parents, Bert and Nicole, for their unconditional support and care during my stay in Delft and abroad.

Tom Gulikers
Delft, December 2018

Contents

List of Figures xi

List of Tables xiii

Nomenclature xv

Abstract xvii

1 Introduction 1

2 Literature Review 3

2.1 Machine Learning with Artificial Neural Networks . . . 4
2.1.1 Definition of a Machine Learning Problem . . . 4
2.1.2 The Artificial Neuron . . . 5
2.1.3 The Artificial Neural Network . . . 7
2.2 Key Elements in Artificial Neural Network Modelling . . . 8
2.2.1 Train and Test Data . . . 8
2.2.2 Activation Functions . . . 9
2.2.3 Cost Functions . . . 11
2.2.4 Artificial Neural Network Training with Backpropagation . . . 12
2.2.5 Tuning Hyperparameters . . . 15
2.3 Employment of Artificial Neural Networks in FEA: State of the Art . . . 19
2.3.1 Direct Modelling of Problem Outputs . . . 20
2.3.2 Constitutive Modelling of Homogeneous Materials . . . 22
2.3.3 Constitutive Modelling of Composite Materials and Laminates . . . 26
2.3.4 Other Data-Driven FE Modelling Techniques . . . 29
2.4 Summary . . . 31
2.5 Discussion . . . 33

3 Constitutive Modelling Framework based on an Artificial Neural Network 35

3.1 Designing the Computational Framework . . . 36
3.1.1 FE Data Generator Module . . . 37
3.1.2 Artificial Neural Network Training Module . . . 39
3.1.3 FE model with Integrated Artificial Neural Network-based Elements . . . 40
3.2 Designing the Artificial Neural Network . . . 41
3.2.1 Baseline Design . . . 41
3.2.2 Including Health Parameters . . . 43
3.2.3 Including Historic Stress-Strain States . . . 44
3.2.4 Minimum Input Design . . . 44
3.3 Computing the Material Jacobian . . . 45
3.3.1 Tangent Stiffness . . . 46
3.3.2 Secant Stiffness . . . 48
3.4 Implementation . . . 48
3.4.1 Modelling Artificial Neural Networks in Python . . . 49
3.4.2 Abaqus UMAT Subroutine in Fortran . . . 49
3.5 Summary . . . 50

4 Creating the Automated FE Data Generator **55**
4.1 Model Creation . . . 55
4.1.1 Choosing the Benchmark Cases . . . 55
4.1.2 Building the FE Models . . . 57
4.1.3 Simulation Settings . . . 58
4.1.4 Post-Processing . . . 58
4.1.5 Scripted Implementation . . . 59
4.2 Material Models . . . 60
4.2.1 High-Strength Steel . . . 61
4.2.2 Carbon-Fibre Reinforced Plastic . . . 61
4.2.3 Damage . . . 62
4.3 Periodic Boundary Conditions . . . 65
4.4 Performance and Convergence . . . 67
4.4.1 Performance and Results . . . 67
4.4.2 Energy Levels . . . 69
4.4.3 Mesh Convergence . . . 70
4.5 Summary . . . 72

5 Analysis and Results **73**
5.1 Modelling an Elasto-Plastic Steel Plate . . . 73
5.1.1 Design of Experiments . . . 73
5.1.2 Hyperparameters and Results . . . 74
5.1.3 Performance and Discussion . . . 76
5.2 Modelling a Composite Plate with an Elliptic Cut-Out . . . 77
5.2.1 Design of Experiments . . . 78
5.2.2 Hyperparameters and Results . . . 78
5.2.3 Comparison of Input/Output Configurations . . . 81
5.2.4 Performance and Discussion . . . 83
5.3 Summary . . . 85

6 Conclusions **87**

7 Future Work **91**

Bibliography **93**

A User Material Subroutine in Fortran **99**

List of Figures

2.1 Sketch of a biological and artificial neuron 6
2.2 Binary classification and logistic regression of a noisy data set 6
2.3 Multi-layer neuron layout 7
2.4 Neural network prediction on sufficiently large and too small training data sets . . 9
2.5 Three-layer ANN with an arbitrary number of inputs, hidden layer nodes and outputs 13
2.6 Influence of learning rate on the convergence of an optimisation algorithm 14
2.7 Influence of a momentum term on an optimisation path 15
2.8 Visualisation of under-fitting and over-fitting 16
2.9 Visualisation of hidden layer activations 16
2.10 Training and testing accuracy during an ANN optimisation 17
2.11 Well-fitted and over-fitted ANN 19
2.12 Optimisation procedure which employs trained ANNs as substitution for FE analyses 21
2.13 Example of a Hopfields Network architecture 21
2.14 FEA solution strategy using a Newton-Rhapson scheme 23
2.15 Problem variable space and neural network architecture 23
2.16 Adapted network architecture to capture cyclic behaviour 24
2.17 Autoprogressive training algorithm 24
2.18 Encapsulated ANN 24
2.19 SelfSim ANN analysis framework 26
2.20 Layout of an ANN which predicts damage size and location from a given strain field 26
2.21 FE modelling approach for laminated composites with an ANN lamina model . . . 27
2.22 Comparison of analytical and ANN failure envelope prediction of a biaxially loaded lamina 28
2.23 Failure envelopes approximated by analytical relations versus an ANN 29
2.24 A data-driven numerical optimisation procedure 30

3.1 Example of substructuring a densely meshed region in a fuselage panel 36
3.2 Schematic representation of the proposed computational framework 37
3.3 Baseline ANN architecture 42
3.4 ANN architecture with health parameters as additional inputs 43
3.5 ANN architecture with historic stress-strain states as additional inputs 44
3.6 ANN architecture with a reduced input vector 45
3.7 Visualisation of tangent versus secant stiffness 46
3.8 Illustration of how the UMAT subroutine fits into the FEA algorithm 50

4.1 Geometry and mesh of the steel plate model 57
4.2 Geometry and mesh of the CFRP plate model 58
4.3 Definition of shear strain 59
4.4 Elasto-plastic response of the high-strength steel material model 61
4.5 Quasi-isotropic layup used for the analyses 62
4.6 Hashin damage evolution model in Abaqus 64
4.7 FEM model of a generalised 3D unit cell 65
4.8 Node-matching in preparation for periodic boundary conditions 66
4.9 Mechanical response of the high-strength steel plate model in biaxial tension . . . 68
4.10 Computational cost for several load cases as a function of mesh refinement 68
4.11 Von Mises stress distribution under a uniaxial load, before and after damage initiation 69

4.12 Visualisation of the crack front subject to a uniaxial load, after damage initiation . 69
4.13 Energy levels in the CFRP model for several loading scenarios 70
4.14 Mesh convergence for uniaxial tension (x-direction) 71
4.15 Mesh convergence for uniaxial tension (y-direction) 71
4.16 Mesh convergence for shear (xy-plane) 71
4.17 Mesh convergence for biaxial tension (x and y-direction) and shear (xy-plane) . . . 72

5.1 ANN response with 1 and 2 HLs for the steel plate model 75
5.2 ANN response for several training times for the steel plate model 75
5.3 ANN response for the steel plate model of two verification cases with the final choice of hyperparameters 77
5.4 Accuracy during the training process of a 2-HL and 3-HL ANN 79
5.5 ANN response with 2 and 3 HLs for the composite plate model 80
5.6 ANN response for several training times for the composite plate model 81
5.7 ANN response of three verification cases for the composite plate model, with the final choice of hyperparameters 82
5.8 Training and testing accuracy during training for several ANN input/output configurations 83
5.9 Comparison of ANN responses for several input and output configurations 84

List of Tables

3.1 Example data set resulting from one 2-D FE simulation 39

4.1 Characteristics of the FE benchmark cases 56
4.2 Hardware specification and analysis settings 58
4.3 Properties of a typical high-strength steel material 61
4.4 Elastic properties of a typical CFRP ply 62
4.5 Ultimate strength of a typical UD CFRP ply and damage parameters 64

5.1 Performance comparison of popular initialisers for the steel plate model 76
5.2 Performance comparison of popular training schemes for the steel plate model . . . 76
5.3 Computational time of each module in the framework for the steel plate model . . 77
5.4 Performance comparison of popular initialisers for the steel plate model 79
5.5 Performance comparison of popular training schemes for the steel plate model . . . 80
5.6 Computational time of each module in the framework for the CFRP plate model . 85

Nomenclature

Abbreviations

Adadelta	Adaptive Gradient Descent with gradients restricted by 'delta'
Adagrad	Adaptive Gradient Descent
Adam	Adaptive Momentum Estimation
ANN	Artificial Neural Network
CFRP	Carbon-Fibre Reinforced Polymer
DoE	Design of Experiments
DoF	Degree of Freedom
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
HL	Hidden Layer
Nadam	Nesterov-accelerated Adaptive Moment Estimation
PBC	Periodic Boundary Conditions
QI	Quasi-Isotropic
RMSprop	Root Mean Square Propagation
RUC	Representative Unit Cell
RVE	Representative Volume Elements
SGD	Stochastic Gradient Descent
SV	State Variable
UD	Unidirectional
UMAT	User Material Subroutine

Symbolic Constants (Structural Analysis)

ζ_{ij}	Health parameter along DoF ij, equal to $\frac{\sigma_{ij}}{\epsilon_{ij}} \cdot \frac{\epsilon_o}{\sigma_0}$	s
C_i	Dimension the model along DoF i	mm
D	Dimension of a model along the z-direction	mm
E_H	Dimension of a cutout along the y-direction	mm
E_W	Dimension of a cutout along the x-direction	mm

H	Dimension of a model along the y-direction	mm
N_{seeds}	Amount of mesh seeds on each outer edge of the FE model	–
t_{inc}	Time increment in a FEA	s
W	Dimension of a model along the x-direction	mm
$\Delta\epsilon_{ij}$	Strain increment in a FE calculation	–
$\Delta\sigma_{ij}$	Stress increment in a FE calculation	$\frac{N}{mm^2}$
ϵ_{ij}	True strain corresponding to DoF ij	–
$\overline{SV}$	State variable vector	–
σ_{ij}	True stress corresponding to DoF ij	$\frac{N}{mm^2}$

Symbolic Constants (Neural Network)

l	Number of an ANN layer, counting from the input layer	–
n	Number of an artificial neuron in an indicated layer	–
N_l	Amount of nodes in HL l	–
N_{batch}	Amount of data points per batch in each ANN training loop	–
N_{epochs}	Amount epochs (iterations) in ANN training	–
$\boldsymbol{w}^l$	Weight matrix for layer l	–
$\bar{b}^l$	bias vector for layer l	–
$\bar{o}^l$	output/activation vector in layer l	–
$\bar{x}^l$	Input vector for layer l, which equals o^{l-1}	–
$\bar{z}^l$	Net input vector in layer l	–
α	Learning rate	–
γ	Decay rate	–
ϕ^l	Activation function for layer l	–

Abstract

Engineering fields such as aerospace rely heavily on the Finite Element Method (FEM) as a modelling tool. In combination with the scale and complexity of the structures typically involved here, computational cost remains a traditional issue. To perform FEM analyses of such structures efficiently nonetheless, engineers rely on techniques such as substructure homogenisation. Essentially, the advantages to using homogenised models are an easier division of labour, less model preparation time and a reduced computational time. Unfortunately, the classical approach to substructuring is either limited to linear elasticity as in the case of static condensation, or is still computationally expensive as non-linear FEA of detailed substructure models need to be performed each time a different loading is applied to the full structure. To improve the efficiency and/or accuracy of homogenised substructures, it would therefore be of interest to develop a methodology which allows to capture a complex structural response without constantly resorting to non-linear FEA.

An emerging technology that may assist in improving the efficiency and accuracy of how homogenised substructures are modelled, is machine learning. While the fundamental principles of this field were developed in the 1940's, the ever-increasing accessibility and magnitude of computational power have resulted in a leap of popularity since the 1990's. Especially the (deep) Artificial Neural Network (ANN), a versatile machine learning framework, has proven to be a promising tool that can perform tasks ranging from image recognition to the failure analysis of composites.

In the current master's thesis, a framework is developed that integrates ANN and FEM techniques as to establish a highly flexible approach to substructure homogenisation. More specifically, this framework allows to establish a homogenised representation of a substructure regardless of its structural complexity (e.g. inclusions or cut-outs) and material complexity (e.g. damage progression and failure). To achieve this, traditional techniques to approximate homogenised behaviour are replaced by a constitutive model that is captured in an ANN.

The developed framework consists of three parts. Firstly, the data generator module creates, runs and post-processes a series of FEM simulations of a chosen substructure, based on a predetermined Design of Experiments. In this DoE, all independent parameters (e.g. applied loads) that influence the response to be modelled should be sufficiently varied. The second module trains an ANN based on the generated data. In doing so, it learns to predict the homogenised mechanical behaviour of the chosen substructure as a function of the independent parameters. The third module then integrates this trained ANN in the FEM software package Abaqus as a user material subroutine (UMAT). The substructure of choice is now homogenised and represented by a single element, which can be readily used together with traditional elements in a global model.

The described methodology was applied to two FEM models of increasing complexity. The first model was a biaxially loaded, 2D elasto-plastic high strength steel material without additional complexities. An initial ANN design was made based on this model and it turned out that a network with 2 Hidden Layers (HL) and 10 nodes per HL was ideal to capture the response. Furthermore, it was determined that the ANN converged after training for 1,500 epochs with the 'Nadam' scheme. The second model was a composite plate with an elliptical cut-out and Hashin damage, thus adding both structural and material complexities. This model was loaded in biaxial tension and in-plane shear. A 2-HL network was found to be the most suitable architecture with 60 and 40 nodes in each HL respectively. A training time of 10,000 epochs was required to reach convergence using the Nadam optimiser, which led to an excellent fit of the mechanical response.

Moreover, several input and output vectors for the ANN were investigated. It was concluded that the best results are obtained if the input vector contains the previous and current stress and strain state as well as the strain increment, whereas the output is the predicted stress increment.

1 Introduction

Background and motivation

Since its introduction halfway the 20^{th} century, the *Finite Element Method* (FEM) has gained an immense amount of popularity as an engineering analysis tool. As of today, engineering fields such as aerospace and automotive rely heavily on FEM to compute the mechanical behaviour of structures under various types of loading. Despite its significant computational expense, engineers have gone as far as to model entire aircraft and other complex structures with this method. However, it is often undesired to perform a FEM analysis of a full-scale model each time when a small component is changed.

Instead of performing analyses on a fully-meshed model, a complex structure may be split into multiple sections based on a method called *substructuring*. Traditionally, this technique combines coarse global models with high-fidelity local substructure models. These substructures are subject to appropriate boundary conditions, obtained from analyses on the global scale. Essentially, there are three advantages to substructuring: an easier division of labour, a reduction in model preparation time and a reduction in computational time.

In order to represent a substructure in the global model, one may condense its structural properties into a single homogenised material model representation [1]. Still, several aspects of these homogenised material models remain to be investigated in order to improve their accuracy, efficiency and applicability [2]. One particular area of potential here is to construct homogenised models of substructures which are capable of capturing complex structural and material responses.

An emerging field that may assist in improving the efficiency and accuracy of how homogenised substructure representations are set up, is *machine learning*. While the fundamental principles of this field date back to the 1940's, the ever-increasing accessibility and magnitude of computational power has resulted in a leap of popularity since the 1990's. Especially the (deep) *Artificial Neural Network* (ANN), which is a versatile machine learning framework, has proven to be a promising tool that can perform tasks ranging from image recognition to the failure analysis of composites. The ANN is essentially an explicit equation with potentially infinite complexity that can be optimised ('trained') to capture any function, given the availability of a sufficiently representative data set.

Aim and scope

The complex modelling capabilities of an ANN, combined with its high accessibility nowadays, provides new opportunities in the field of structural analysis. Engineers have already applied ANNs to predict buckling loads of various types of structures, as well as specific material properties and simple mechanical responses.
The current master's thesis aims to develop a framework that integrates ANN and FEM techniques, as to provide a novel approach to homogenised substructure modelling. More specifically, this work strives to establish a procedure which allows to construct a homogenised representation of any substructure regardless of the structural complexity (e.g. inclusions or cut-outs) and material complexity (e.g. damage progression and failure). To achieve this, traditional techniques to obtain the homogenised behaviour of structural sections are replaced by a constitutive model that is captured in an ANN. With this in mind, the main research objective for this thesis is identified as follows:

The objective within the time-span of this thesis is to improve the accuracy and/or computational efficiency of substructuring in FEM by implementing a constitutive model based on an artificial neural network, to be trained on a detailed and representative data set.

To achieve this goal, a set of four research questions are formulated to guide and structure the research. The first one addresses background knowledge on ANNs, whereas the remaining questions focus on the actual research topic.

I What artificial neural network set-ups, architectures and training procedures are suitable to model constitutive behaviour? Furthermore, how can a sufficiently representative training data set be constructed from a minimum amount of simulations?

II To what extent can an artificial neural network be used to capture complex homogenised behaviour of a substructure? Furthermore, what are the conditions such that a substructure is fit for usage with the proposed method?

III Is it feasible to integrate an artificial neural network into a finite element analysis procedure as a constitutive model?

IV If the previous question is answered positively, what are the gains in computational time and/or accuracy from using the artificial neural network-based set-up?

Thesis layout

The current work starts with a literature review in Chapter 2, where the reader may find a background knowledge and reference on machine learning with ANNs. Furthermore, this chapter elaborates on the state of the art pertaining to the application of ANNs in structural modelling. It concludes with a reflection on the found information and the substantiation of the research gap in which this work fits. Based on the findings from the literature review, Chapter 3 describes the development and set-up of the framework to construct and use an ANN-based constitutive model. This framework consists of three modules: the data generator, the ANN training procedure and the actual implementation of the model in FEM. In order to apply the proposed framework, Chapter 4 describes the set-up of two FEM models, which are chosen to set up an initial configuration of the ANN and demonstrate the framework's capabilities respectively. More specifically, this chapter elaborates on the material models and boundary conditions used and evaluates the convergence and performance of these models. The findings of the previous chapters come together in Chapter 5. Here, the ANN is configured and fine-tuned for both FEM models and the ANN's performance and capability to capture their mechanical behaviour is analysed. To conclude the current thesis, the research questions are answered in Chapter 6 whereas Chapter 7 presents the author's thoughts on how this work may be continued.

2

Literature Review

As motivated in the introduction, the current thesis aims to combine the fields of machine learning and structural analysis. The procedure proposed in this work is designed based on a literature review on two main subjects.

Firstly, the theory behind machine learning with Artificial Neural Networks (ANNs) is studied as well as the key aspects of effectively training and employing this tool. To guide this study, the following literature review questions are set up.

- How does one set up a machine learning problem and what are the required ingredients?
- What is an artificial neuron and how is it used to build an ANN?
- To what extent can ANNs learn patterns; Are there limits?
- What happens during the training/optimisation process?
- How does one generate a suitable data set for training and testing purposes?
- What methods exist to evaluate and improve the performance of an ANN?

Furthermore, the state of the art literature on the subject of ANNs applied in structural analysis and the Finite Element Method (FEM) is reviewed. Regarding this topic, the following questions are studied.

- To what extent and benefit are ANNs usable as a global substitute for an FE model?
- What are the current applications and benefits of employing an ANN to capture constitutive behaviour?
- What techniques, architectures and training algorithms are used to represent a constitutive model by an ANN?

Following the presented literature review questions, this chapter first provides a general background on machine learning, artificial neurons and ANNs in Section 2.1. Subsequently, Section 2.2 presents a more in-depth look at the techniques and required knowledge to program and properly train an ANN. The literature concerning the application of ANNs in the field of structural analysis and specifically FEM is then reviewed in Section 2.3. Finally, the literature review questions are answered and a motivation for the research objective of the current thesis is presented in Section 2.5.

2.1. Machine Learning with Artificial Neural Networks

Machine learning is a broad term that includes all algorithms which are able to extract patterns from a data set. In order to apply such techniques, it's essential to have an understanding of how machine learning works and what the advantages and drawbacks of certain choices in a set-up may be.

The goal of the current section is to establish an overview and summary of key features of machine learning with an ANN. Firstly, the fundamental set-up of a machine learning problem is addressed in Subsection 2.1.1. Then, the motivation behind the artificial neuron and its mathematical description are elaborated on in Subsection 2.1.2. Finally, a description of the ANN is provided in Subsection 2.1.3.

2.1.1. Definition of a Machine Learning Problem

This subsection aims to provide an intuitive definition of a machine learning problem, which is the fundamental concept behind self-learning computational units such as ANNs. Each problem consists of three essential components which are: the task, the performance measure and the experience, as explained by Goodfellow et al. [3]. Mitchell [4] motivates the presence of a natural relation between these entities: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Task When a computational unit (such as an ANN) performs a task, this refers to the process that computes the output corresponding to a given input (called 'example'). The learning process itself is not a task, but is rather defined as the development of the computational unit such that it can perform the desired task with good performance [3].

A good overview of possible tasks is defined in the 5^{th} chapter of Goodfellow et al. [3]. The two most commonly encountered ones are summarised below [5].

- *Regression:* The most basic technique to create a trend line based on a data set is to apply regression. The output of a regression task is always a function that approximates the labels of a data set. Thus, the resultant function predicts an output value based on n input values: $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. This function is not necessarily a straight line and may include many trainable parameters to represent a complex non-linear problem. Regression is therefore not only the most basic, but also the most general task an algorithm may have [6]. A more challenging task is to perform a regression on data sets where some data points are missing. This situation requires that multiple instead of one function will be the output of the task; one for each subset of potentially missing data points.
- *Classification:* This type of task is similar to regression, but instead of producing a continuous range of values it returns a prediction of which class an input example belongs to. The simplest case is binary classification, where each label is either a zero or one. A well-known example within the field of ANNs is the prediction that a cat is present in an input picture, based on a labeled training data set[1]. A more advanced form of classification is the multiclass alternative, which performs the same action with a finite number of classes. In practice, this often translates to a number of outputs from a task equal to the number of possible classes. Each output then provides the probability that its corresponding class is true, from which the best fit is selected [5]. The classification of a set of n input values to k classes is formally defined by the function: $f : \mathbb{R}^n \longrightarrow 1, ..., k$

Performance measure In order to facilitate a machine learning process, the performance of a computational unit has to be evaluated quantitatively [3]. This performance measure is intuitively

[1] `https://www.datasciencecentral.com/profiles/blogs/dogs-vs-cats-image-classification-with-deep-learning-using`, accessed 19-02-2018

related to the performed task and always a function of the offset (error) between predicted and true data labels in a training or testing data set.

The type of performance measurement is to be chosen by the user. It is a critical part of the machine learning problem design which defines how certain types and magnitudes of errors translate into a learning step. For problems with continuous outputs, it is common to use a cost function such as the ones described in Subsection 2.2.3. The choice should always be made with respect to a specific problem, as it is often desired to treat each output error differently. For example, it may be so that an error value above a certain threshold for output a is considered unacceptable and heavily penalised (the learning process then puts 'extra effort' in reducing this error). In the same system, output b may be less critical and feature a simple linear or logarithmic penalty [3].

Another method of measuring performance is to quantify the total accuracy of a trained computational unit on a data set. As explained in Subsection 2.2.1, labeled data is always divided in at least a training and a testing data set in the context of machine learning and neural networks. Measuring the accuracy of the trained computational unit on these sets serves as an indicator of how well the labels are predicted. Based on these values, one can make adjustments to the hyper-parameters (settings of the computational unit and training process) if necessary. For example, a low training and testing accuracy can be an indicator that the training time is too short or a lack of complexity in the model. A low testing accuracy in combination with a high training accuracy may be due to over-fitting of noisy training data (covered in Subsection 2.2.5).

Experience Experience is the collection of variables that characterises a computational unit, based on the data points it has been trained on and the training process itself. As explained in Goodfellow et al. [3], supervised learning is a specific type of learning where a training data set is used that contains labels (target outputs). Another type of learning is unsupervised learning, which is an algorithm that learns properties or characteristics of a certain data set without a predefined output (such as a probability distribution). In the case of ANNs, supervised learning is the method of choice.

A supervised learning algorithm builds its experience based on the task and performance described previously. From this point, the learning process has become a typical optimisation problem which can be approached in many different ways. Each optimisation method takes "steps" by modifying the system variables (building experience) based on evaluations of the cost function (performance measure). In the specific case of ANNs, it is common to use the gradient descent method by computing the derivatives of the cost function to each system variable, as explained in Subsection 2.2.4.

2.1.2. The Artificial Neuron

Each human possesses an incredibly capable non-linear modelling tool. This biological neural network, also known as 'the brain', consists of about 100 billion biological neurons which each perform a simple operation on electrical signals it receives from other neurons. Together these neurons are capable of high-frequency image recognition and calculations. Furthermore the brain is able to make predictions, ranging from human emotions to trends in the global financial markets, albeit with debatable accuracy.

As the name suggests, the fundamentals of artificial neurons (also called 'nodes') and ANNs lie in theories based on how neurons in the human brain operate and cooperate. Whilst the processes inside the brain are still not fully understood, scientists all over the world have been intrigued by its capability to understand and predict even the most non-linear relations in data [7]. In the endeavour to create a model that is capable of (partially) mimicking the behaviour of a biological neural network, the first neuron-like computational unit was proposed by McCulloch and Pitts [8]. This model had the simple functionality of performing logical threshold operations based on a set of inputs, as schematically shown in Figure 2.1. It was based on the mathematical notion of computation by Alan Turing, which still serves as a fundamental building block in creating computing machines.

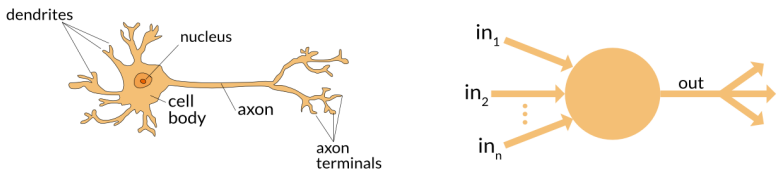
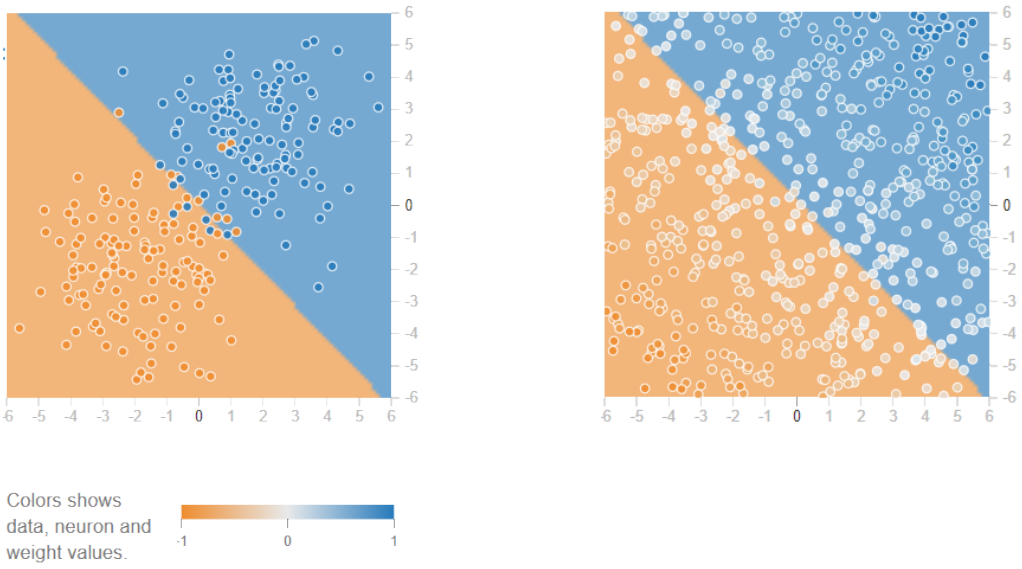


Figure 2.1: Sketch of a biological and artificial neuron [2]

Single-layer neuron One of the first practical application of machine learning was demonstrated by Hebb [9], who invented the Hebb synapse; a function that operates on a set of input data by mathematical operations to fit a predetermined output. This so-called single-layer neuron or node is a simple linear artificial neuron which maps an input vector $\bar{x}$ to an output vector $f(\bar{x})$ by multiplication with a weight vector. The functionality of a single-layer node is confined to simple problems which classify outputs into two linearly separable classes, as it is not possible to capture non-linear behaviour with a linear operator [10]. In other words, this node predicts to which class a set of input data leads and thereby functions as a classifier. The output is a value between 0 and 1 which indicates the likelihood that the set of inputs leads to class A (closer to 0) or B (closer to 1), which is called logistic regression. Often, an extra operation is performed that rounds all values below 0.5 to 0 and above 0.5 to 1, which is called binary logistic regression. A visualisation of (binary) logistic regression is shown in Figure 2.2.



(a) (b)

Figure 2.2: Binary classification (a) and logistic regression (b) of a noisy data set [11]

The mathematical definition of a node designed to perform binary logistic regression is as follows.

$$f(\bar{x}) = \begin{cases} 1, & \text{if } \overline{w}^T \bar{x} + b \geq \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

In this equation, the weight vector $\overline{w}$ and bias b are optimised based on available training data, as elaborated on further in Subsection 2.2.1. Essentially, an algorithm passes each entry in a training data set through Equation (2.1) and checks if the output is correct. If not, the weights and biases are increased or decreased depending on the observed error.

[2] `https://appliedgo.net/perceptron/`

Although the applications of a Hebb synapse are limited by the linearity condition, the idea of 'training' a computational unit was born.

Multi-layer neuron The single-layer model proposed by Hebb was extended by Rosenblatt [10] to form the very first multi-layer artificial neuron. Rosenblatt's design was not linear and consisted of two sequential mathematical operators. In this multi-layer unit, $\overline{w}^T \bar{x} + b$ was called the 'net input' z of the node and was fed into an 'activation function' ϕ, as schematically shown in Figure 2.3. The output o of this activation function formed the output of this multi-layer node.

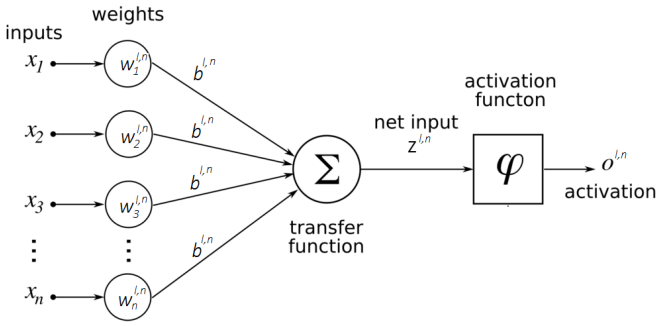


Figure 2.3: Multi-layer neuron layout with inputs $\bar{x}$, weight vector $\overline{w}$, bias b and activation function ϕ [3]

2.1.3. The Artificial Neural Network

To be able to characterise complex patterns in a given data set, it is intuitive that the use of a single node is not sufficient. Multiple artificial neurons can therefore be combined in an Artificial Neural Network (ANN), featuring multiple layers and nodes per layer. Extra layers between the input and output layers are called Hidden Layers (HL), as the values in these layers are not monitored. The complexity of the ANN architecture directly relates to the complexity of problems that can be modelled with it. For example, one can employ a complex neural network to accurately and efficiently perform image recognition tasks [12]. For other tasks, such as the recognition of erroneous data in a small data set, a simple network may suffice. For any complex problem, a network of artificial neurons may be established with multiple parallel nodes per layer and one or more layers. These networks feature the following five fundamental aspects, as described by Rumelhart et al. [13]:

1. Multiple processing units or artificial neurons;
2. For each layer l, the state (or net input) vector $\bar{z}^l$ is the linear multiplication $\boldsymbol{w}^l \bar{x}^l + \bar{b}^l$;
3. Each ANN layer maps $\bar{z}^l$ to the output vector $\bar{o}^l$ through an activation function ϕ^l (also known as 'transfer function') which computes the nodal output from a given input by applying a non-linear mapping to the net input;
4. A definition of which neurons are connected in a network, and how;
5. The learning rule, which defines how the weights and offsets are modified based on the observed error.

In order to train an ANN properly, the correct outputs (labels) corresponding to a series of inputs are predetermined for a sufficiently large and representative data set. The weight matrices and biases of the ANN are then given an initial value and the series of inputs are passed through the network. An error function is then employed on the ANN output to measure the performance

[3] `https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png`, accessed 06-02-2018

for one or multiple inputs (batches), based on which the weight matrices and bias vectors are adjusted. After training the network, it functions as a predictor for inputs that are similar to the training data. Setting up and effectively training an ANN requires a basic understanding of these fundamental concepts, which are described in more detail in the next section.

2.2. Key Elements in Artificial Neural Network Modelling

This section focuses on some of the essential concepts used to train machine learning systems, with a particular focus on ANNs.

Firstly, some general information about train and test data is provided in Subsection 2.2.1. Afterwards, several activation functions with their respective benefits/drawbacks are discussed in Subsection 2.2.2. Subsection 2.2.3 covers the basics of cost functions, which are a necessary ingredient to train an ANN. Then, the most common training algorithm is covered in Subsection 2.2.4: Gradient descent based on backpropagation. Finally, some guidelines on choosing hyperparameters to counter common training issues, such as over-fitting and under-fitting, are covered in Subsection 2.2.5.

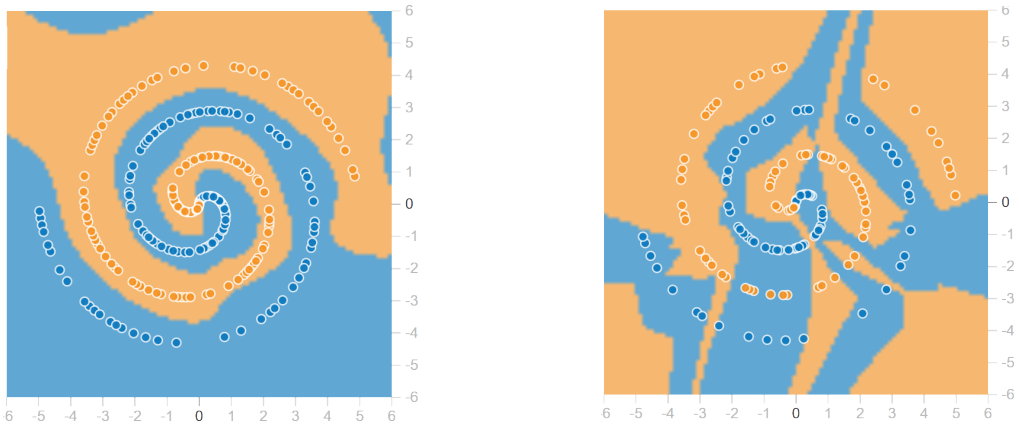
2.2.1. Train and Test Data

An ANN can only perform well once it has been trained on a proper data set. The choices made by the engineer concerning the data are one of the most critical aspects of building a well-functioning ANN [14]. As explained in Subsection 2.2.4, the network is 'trained' by optimising weights and offsets for each node, such that the overall prediction error of the complete ANN is minimised. The cost function represents this magnitude of error between the network outputs and the labels of the data (expected outputs), as discussed in more detail in Subsection 2.2.3.

To train an ANN, one therefore needs a sufficiently large and representative data set. This data should be 'labeled', meaning that the result (output) of each data point is known. Typically, the available data is divided into at least two subsets: the training and testing set [12]. The training set is often the largest in volume and serves to train the network, while the test set serves to provide a measure of how good the trained network performs. This is done by computing the error of the test data outputs from the trained network. There is no fixed rule in how to choose and divide the data into sets, but mostly comes down to experience or trial and error. Common practice is to employ a training/testing division of 80%/20% or 90%/10%. In the coming paragraphs, some of the most common issues related to training and test data sets are covered based on the works of Goodfellow et al. [3], Nielsen [5] and Ng [14].

Data set volume If the training data set is not sufficiently representative to define the problem's behaviour, the test error is often the first indicator that there is an issue by returning a (significantly) larger error magnitude than the training error. A visual example is shown in Figure 2.4, where an ANN is trained twice on the same pattern. The amount of training data is different for both, however, and it is observed that the network shown in Figure 2.4b is not capable of accurately predicting the double spiral shape. The main demerit is that the spacing between training data points is too large such that the network incorrectly allows a false classification of these gap regions. The result is a low training error for both cases, while the test set error is about 20 times higher for the case shown in Figure 2.4b as compared to Figure 2.4a.

Data domain Each data set has a range of input variables that it covers, which is called the domain. Typically, one should be careful that the domain of the training data set is sufficiently large and diverse, such that no predictions outside the training domain are made. An insufficiently large training domain may manifest itself as the combination of a low training error and a high testing error.



(a) (b)

Figure 2.4: Neural network prediction on sufficiently large (a) and too small (b) training data sets

2.2.2. Activation Functions

Each node in an ANN has an activation function, which is applied to the net input z as shown in Figure 2.3. This function most often has a non-linear shape and produces the 'activation' or output o, which is the value that is passed to the next layer of neurons. There are multiple reasons why it may be desirable to implement activation functions in an ANN. Each function has advantages and disadvantages and it is up to the user to select a proper function for each neuron in the network [3].

When assigning an activation function, the following properties have to be considered.

- Non-linearity: It is desirable to use a non-linear activation function, as it enables the ANN to generalise on non-linear relations in data. While a purely linear neuron only models a linear trend, a non-linear neuron can capture a parabolic or tangential trend [5]. In fact, it can be proven through linear algebra that a network consisting of only linear activation functions can always be reduced to a single neuron which means that such a network can never represent non-linear trends. In contrast, the Universal Approximation Theorem by Cybenko [15] proves that an ANN with a single HL and non-linear activation function is a universal approximator for any non-linear relationship.
- Range: Depending on the case, an activation function can map the net input in a neuron to a certain range such as <0, 1>, <-1, 1> or [0, ∞>. An activation function with a limited output range is often desirable for the output layer of a network, as it improves the stability of most optimisation algorithms that employ a gradient based backpropagation approach. In the HLs of an ANN, this is not a desired property as a limited output range tends to slow down the learning process [6].
- Derivative: In order to train an ANN using backpropagation, the derivatives of the chosen activation functions are important. The learning speed for such methods is directly proportional to the derivative at a certain activation value, thus a function with a horizontal asymptote implies an increased risk of slow training progression [5, 6].
- Function value near origin: It is desirable, but not required, to use an activation function which approaches the identity function near the origin. It turns out that an ANN can be initialised more effectively when activation functions have this property, as explained by Sussillo [16]. One can assign initial weights and offsets which are random, but close to zero and therewith ensure a large initial learning speed.

With those aspects in mind, the following paragraphs describe the most commonly used activation

functions.

Identity

$$I(z) = z$$

The identity function is a linear function with range $<-\infty, \infty>$ and a constant derivative equal to 1. Its output is obviously equal to its input and therefore, a node with this function is simply a linear regression tool. This activation function is not suitable to be used in ANNs which are trained on data with non-linear trends.

Binary step (heaviside)

$$H(z) = \begin{cases} 1, & \text{for } z \geq 0 \\ 0, & \text{for } z < 0 \end{cases}$$

This function is used for binary classification and when a neuron uses it as activation function, this neuron is called a perceptron. Its range is obviously 0, 1 and the derivative is always zero (except at x=0, where it is undefined). Because of this property, it is not practical to combine the binary step function with backpropagation and the function is not often seen in ANNs [5].

Sigmoid

$$Sigmoid(z) = \frac{e^z}{1 + e^z}$$

Historically, the sigmoid is the famous activation function and has been widely used by machine learning engineers. It always produces an output in the range <0, 1> and has a continuous, non-zero derivative. In contrast to the binary step function, the sigmoid is suitable to be used in ANN training because of its non-zero derivative. However, in many cases it is applied to achieve the same goal as the binary step, which is binary classification [5]. When the ANN is trained using the benefits of the sigmoid, a binary conversion is performed which maps the final network outputs above 0.5 to 1 and all final outputs below 0.5 to 0. This is, however, not the only use of the sigmoid function and it is straight-forward to generalise to a multi-layer classification approach (for example, the SoftMax function)

Because the output of the sigmoid is limited to values between 0 and 1, this function is mainly used in the output layer of an ANN. This way, the final output is normalised and directly represents a probability that any of the output classes is true.

Hyperbolic tangent

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

This function produces an output in <-1, 1> and is a scaled version of the sigmoid, used for similar applications. Interestingly, the hyperbolic tangent turns out to be a better choice than sigmoid in almost every case. The reason for this is that the mean of the activation is closer to zero and the range is twice as large, which makes learning faster by featuring a higher derivative. However, the same issue as with the sigmoid function is still observed: If the input becomes large, the slope approaches zero and learning is slowed down. The tanh function is mostly used as activation for the output layer, for the same reason as the sigmoid is.

ReLU

$$\text{ReLU}(z) = max(0, z)$$

The Rectified Linear Unit (ReLU) is one of the most popular activation functions used in ANN designs. This non-linear function is a rectifier with an output range of $[0, \infty>$ and a well defined derivative (except at z=0). If the net input of a node is lower than zero, the ReLU function always results in zero. For a net input larger than zero, the output is the identity function. In the region where the derivative is equal to one, learning is fast when using backpropagation optimisation

which is the main benefit of the ReLU function. If the output is zero, however, a ReLU neuron is said to be deactivated and does not participate in the training process as long as it is in this state. The sigmoid and tanh functions actually experience the same kind of issue when z is either very large or very small, but they always maintain a small gradient and thus a capability to participate in backpropagation. In practice, there are generally enough neurons that feature net inputs above zero in a network such that the advantages of ReLU outweigh the disadvantages [5, 6].

Softmax

$$\text{SoftMax}_i(\bar{z}) = \frac{e^{z_i}}{\sum_{n=1}^{N} e^{z_n}} \qquad \text{for i = 1, ... , N}$$

The SoftMax function combines multiple advantages at once. In contrast to other activation functions, the SoftMax takes the entire net input vector of all N nodes as an input. It then computes the exponent of each component i and maps the new vector to values in <0, 1>, similar to the sigmoid function. However, the output vector that results from the SoftMax has the additional property that the sum of all components is equal to one. The output is readily usable as a multi-layer classifier as it indicates a probability of each class (output) being true. Similar to the Sigmoid and Tanh, the SoftMax function is not often used in the hidden layers but rather in the output layer.

2.2.3. Cost Functions

In the previous subsections, it is pointed out that a set of ANN parameters is to be optimised for it to learn patterns. As with each optimisation process, this means that some form of cost (error) function must exist. The purpose of this cost function is to measure the ANN output accuracy $\hat{y} - y$ corresponding to a labeled input x, when the true label is y. A cost function is chosen by the user and can take different forms, but have to abide to the following two rules according to Nielsen [5] and Papalambros and Wilde [17].

- The cost function should be written as an average of the individual costs of the n training examples considered. By enforcing this rule, the cost function always produces a scalar such that it is possible to compute the derivative of this function with respect to the weights and biases. This is a necessary property to use backpropagation for the optimisation process, as will become clear in Subsection 2.2.4.
- The cost function should be formulated as a function of the ANN output(s) only. With this approach, input and hidden layers remain untouched in the cost function computation. As with the previous rule, the reason for this restriction is to allow backpropagation, which would not be possible if the cost function depends on other network layers than the output layer.

For the specific application in ANNs, some of the most common cost functions C are elaborated on below. In these functions, y^i and $\hat{y}^i$ are the true and predicted label respectively, corresponding to the i^{th} data point and n is the total number of considered data points in one batch. While it is recognised that many more cost functions exist beside the four stated below, these are not required for the current research as only relatively simple ANNs are employed.

Mean squared

$$C = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

This quadratic error indicator is mainly fit for regression problems. The equation is widely used in statistics and can be used to draw a line which features a minimum average squared distance between it and each data point. In the application of ANNs, this function generally does not work well as it tends to cause slow learning progression in combination with certain activation functions that have horizontal asymptotes (such as the sigmoid). It is therefore only suitable

for performing logistic regression. Furthermore, the mean squared function only depends on the absolute difference, meaning that two large numbers which are relatively close to each other still result in a high loss.

Mean squared logarithmic

$$C = \frac{1}{n}\sum_{i=1}^{n}(\log(y^i + 1) - \log(\hat{y}^i + 1))^2$$

This function differs from its non-logarithmic counterpart in the sense that it has a different sensitivity to the variance of the actual an predicted data labels. In contrast to the mean squared function, two large numbers that lie close together produce a negligible loss value. If the numbers are small, there is barely any difference between the regular and logarithmic mean squared loss so the latter basically provides a way of correcting the large number incompatibility. Similar to the mean squared approach, this function is used mainly in logistic regression.

Mean absolute

$$C = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)$$

This is a simple linear error indicator that is mainly fit for linear regression. In the application of ANNs, this function generally does not work well as it tends to cause slow learning progression in combination with certain activation functions that have horizontal asymptotes (such as Sigmoid). It is therefore mostly suitable for performing logistic regression. Furthermore, the Mean Absolute function only depends on the absolute difference, meaning that two large numbers which are relatively close to each other still result in a high loss. However, this function is still more robust to outliers compared to Mean Squared as larger errors are not amplified.

Cross entropy

$$C = -\frac{1}{n}\sum_{i=1}^{n}\left[y^i \log(\hat{y}^i) - (1 - y^i)\log(1 - \hat{y}^i)\right]$$

This function provides a measure of divergence when considering two probability distributions: the log-likelihood of label $\hat{y}^i$ to be true compared to the actual model output y^i. It is mostly used as loss function in binary classification (although it can be generalised to multi-class applications) and has some major advantages over classical mean squared functions. When using activation functions with horizontal asymptotes, the cross entropy loss will not cause a slow learning rate. Rather, the learning rate is intuitively related to the error magnitude. Therefore, the cross entropy function generally causes fast and controlled (i.e. no overshoot) convergence.

2.2.4. Artificial Neural Network Training with Backpropagation

One could describe the ANN as a predefined function which can take any shape, depending on the choice of the coefficients in this function. For an ANN, these coefficients are the weights and biases and are also called the training parameters. In order to correctly shape this function, the weights and biases have to be optimised by performing a so-called training process. By doing this correctly, one is able to change the training parameters such that the network is able to predict a desired output vector from a given input based on the trends it has extracted from a data set.

This subsection describes the most popular scheme that can be used to efficiently train ANNs: gradient descent with backpropagation. While other algorithms exist, these are generally specifically used for complex and non-convex optimisations which is beyond the scope and requirements of this thesis.

Gradient descent algorithms are a widely used class of optimisers which are able to find local optima of a cost function [17]. These algorithms, much like the name suggests, require the gradients

(derivatives) of the cost function with respect to the problem variables at each increment to determine the step size and direction. However, these gradients are not always available or may be expensive to compute. In the context of ANNs, it turns out that there's an efficient, analytical approach to calculate the gradients. The method is called backpropagation and is one of the main reasons why ANN training can be achieved efficiently. The general formulation of the gradient descent approach based on a Taylor expansion is formally stated as follows [17].

$$\begin{aligned} f(\bar{x} + h\boldsymbol{s}) &= f(\bar{x}) + (\nabla f)^T \bar{s} h + o(h^2) \\ \Longrightarrow df &= f(\bar{x} + h\bar{s}) - f(\bar{x}) \approx (\nabla f)^T \bar{s} h \\ \text{Optimal step direction: } \bar{s} &= \nabla f \end{aligned} \tag{2.2}$$

Where f represents the cost function (ANN output, in this case), $\bar{x}$ is the input vector, h is a step size and $\bar{s}$ is a vector containing the derivatives of the cost function with respect to each variable.

The layout of a single neuron has already been covered in Subsection 2.1.2. In this section, an ANN with one hidden layer as shown in Figure 2.5 will be used to illustrate the gradient descent algorithm. The inputs are $\bar{x}$ and the nodes are indicated with I, H and O for the input, hidden and output layer respectively.

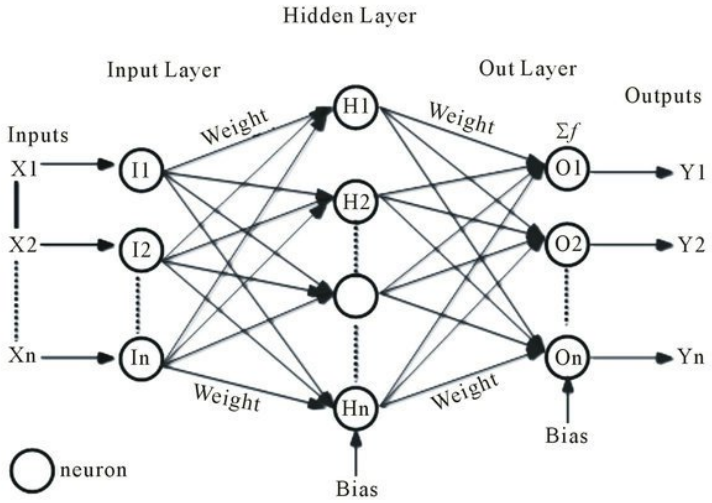


Figure 2.5: Three-layer ANN with an arbitrary number of inputs, hidden layer nodes and outputs

The optimisation variables are $\boldsymbol{w}^l$ and $\bar{b}^l$. These quantities have to be chosen such that the cost function function is minimised. The method used here to arrive at this solution is gradient descent optimisation using backpropagation. Each ANN features at least one input layer, which propagates a set of values through one or multiple hidden layers after which the output layer forms the final network output. After this step, backpropagation helps to determine which weights to modify by computing the derivatives of the cost function with respect to the inputs. This is possible because the transfer function of each node is known, as demonstrated below.

The derivative of the cost function with respect to an ANN variable can be viewed as a quantity which is equivalent to the error in that variable and can be written as Equation (2.3) [5]. Thus, $\delta^{n,l}$ is a measure for the error in node n of layer l.

$$\delta^{n,l} = \frac{\partial C}{\partial z^{l,n}} \tag{2.3}$$

Once this equation is evaluated for every parameter $z^{l,n}$, the optimal step direction $\boldsymbol{s}$ from Equation (2.2) is found and progression towards minimising the cost function can be made. The basic principle used to evaluate Equation (2.3) is based on performing a reverse computation in the

network, from output to input [5]. In order to work in reverse, it is essential to be able to describe the error of a neuron in a chosen layer l in terms of the errors in the next layer $l+1$.

To understand this process, let us first consider forward propagation for layer l. This is the calculation performed on the output of the previous layer to get to the output of the current layer, according to Equation (2.4). This is the regular calculation procedure in a multi-layer node. Here, $\bar{o}$ is the activation, which is the output of activation function ϕ and consequently, the artificial neuron.

$$\begin{aligned} \text{input: } & \bar{o}^{l-1}, \boldsymbol{w}^l, \bar{b}^l \\ & \bar{z}^l = \boldsymbol{w}^l . \bar{o}^{l-1} + \bar{b}^l \\ & \bar{o}^l = \phi^l(\bar{z}^l) \\ \text{output: } & \bar{o}^l \end{aligned} \tag{2.4}$$

Once the forward propagation calculation of the whole network has been performed, the final output value or vector $\bar{o}^{out}$ fed into the cost function. Then, the derivatives of the cost function C to each variable $\frac{\partial C}{\partial \boldsymbol{W}^l}$ and $\frac{\partial C}{\partial \bar{b}^l}$ are computed which is a slightly modified way of writing Equation (2.3). In the interest of compact notations, these two quantities will be written as $d\boldsymbol{W}^l$ and $d\bar{b}^l$ respectively. The backward propagation calculation for layer l is defined as in Equation (2.5), as demonstrated in Nielsen [5].

$$\begin{aligned} \text{input: } & d\bar{o}^l \\ & d\bar{z}^l = d\bar{o}^l * \phi'^l(\bar{z}^l) = \bar{\delta}^l \\ & d\bar{o}^{l-1} = \boldsymbol{W}^{lT} d\bar{z}^l \\ & d\boldsymbol{W}^l = d\bar{z}^l o^{l-1} \\ & d\bar{b}^l = d\bar{z}^l \\ \text{output: } & d\bar{o}^{l-1}, d\boldsymbol{W}^l, d\bar{b}^l \end{aligned} \tag{2.5}$$

The backpropagation procedure described above is performed on the whole network and results in a set of matrices $d\boldsymbol{w}^l$ and vectors $d\bar{b}^l$. These represent the direction in which the gradient of the cost function is steepest and can therefore be used to update the weights and offsets such that the cost (error) is reduced, according to Equation (2.2).

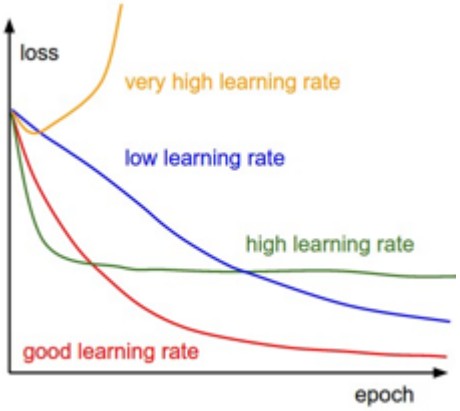


Figure 2.6: Influence of learning rate on the convergence of an optimisation algorithm

While the step direction is computed now, the magnitude or learning rate should be chosen in order to take a suitable step. This quantity is of significant importance to the speed and quality of

convergence, as shown in Figure 2.6. It is common practice that the user defines an initial learning rate α, which can be adapted as the optimisation progresses.

Even with the perfect choice for the learning rate, the optimisation process can be improved significantly by using a momentum term [18]. Roughly speaking, this term increases the learning rate in the direction that is repeatedly part of the optimal direction. An illustration of such a momentum term is visualised in Figure 2.7, where it is clearly seen how the zig-zag optimisation pattern is improved by the momentum approach. The use of a momentum term does have negative

(a) (b)

Figure 2.7: An optimisation path without (a) and with (b) the use of a momentum term

implications as well, however. The most straight-forward backpropagation optimisers based on gradient descent may only require the learning rate and decay rate as user-defined settings. In contrast, a more advanced version algorithm such as the Adam optimiser requires at least two more variables, namely β_1 and β_2. These quantities can regulate how strongly different moving averages in the more complex algorithm contribute to the optimal step direction and magnitude. While such an optimiser may be capable of faster convergence, it does require the user to find the optimum values for at least two additional parameters for a specific ANN model.

2.2.5. Tuning Hyperparameters

A difficulty in the employment of ANNs is the amount of hyperparameters, which is the collective term indicating all settings the user has to define before commencing the ANN training process. The following list provides the reader with an indication of what these hyperparameters specifically encompass.

- ANN architecture: The amount of hidden layers, neurons per layer, activation functions;
- Data: Training / testing ratio, normalisation parameters, cross-validation method;
- Optimisation: Learning rate α, decay rate λ, (β_1, β_2, momentum), amount of epochs, data points per batch, initialisation of weights and biases.

One can imagine that it is extremely difficult to simultaneously optimise all these quantities, assuming that an optimal combination exists. Most of these hyperparameters are heavily correlated, such that the problem of optimising them may be a more complex one than the function that the ANN is trying to capture. In general, getting close to the optimal combination of hyperparameters requires an experienced engineer [14]. Without this experience, one has to follow to guidelines, experience or resort to optimisation algorithms such as grid searching or random searching, which are quite costly approaches.

When tuning the hyperparameters of a machine learning problem, a good understanding of the terms bias (under-fitting) and variance (over-fitting) is required [5, 14]. The meaning of these terms is intuitively shown in Figure 2.8 from which it is seen how both can result in a poorly performing ANN. Furthermore, it is required to understand the difference and relation between training and testing accuracy and how certain hyperparameter choices effect them. In this subsection, a compact overview is provided of the most common principles and guidelines used to obtain a good fit. This subsection elaborates on some of the essential aspects in training an ANN.

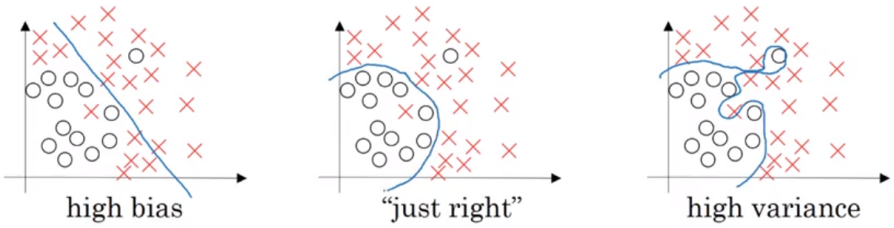


Figure 2.8: Visualisation of under-fitting (bias) and over-fitting (variance) [14]

Amount of hidden layers and nodes per layer The choice of network architecture and activation functions generally corresponds for the degree of complexity that can be modelled with an ANN. While it has been proven by Cybenko [15] that a one-layer ANN can approximate any non-linear function, it is desirable to investigate the performance of multi-layer architectures.

First of all, it is important to recognise that the exact meaning of having multiple HLs is not fully understood [5]. While it is intuitive that a linearly separable model does not require any HLs, it is difficult to reason how many are ideal for a model with multiple non-linear trends. Perhaps the best explanation is found when applying ANNs to a classification problem as shown in Figure 2.9. When visualising the operations of each HL (effectively 'revealing' them), it is seen that each next HL combines the features of the previous layer into more complex shapes. The first HL may only recognise straight lines in several orientations while the second HL combines these lines into triangles or rectangles. Extrapolating this reasoning, the final HL may combine all intermediate features to recognise a specific pattern [19].

It is important to recognise that more is not always better, as shown by the works of Fahlman and Lebiere [20] and Nakama [21]. One may find that a higher accuracy is achieved with a 2-HL network with 4 nodes per layer, than with a 1-HL network that has 50 nodes. A 4-HL network with 10 nodes per layer may perform worse again, thus suggesting the existence of a local optimum. Therefore, determining the amount of HLs and nodes per layer is an iterative procedure and one should start investigating a 1-HL network and gradually increase the amount of nodes. Then, increase the amount of HLs and start over with a small number of nodes until an optimum is found.

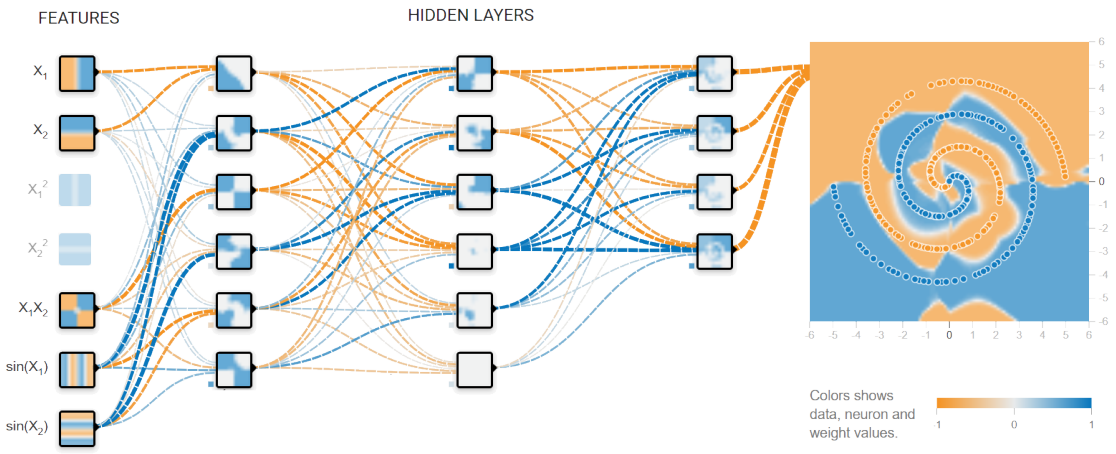


Figure 2.9: Visualisation of hidden layer activations, which show how a set of features (inputs) are processed to come to a certain output [11]

Data normalisation In theory, it is not necessary to perform normalisation on ANN input data as any re-scaling operation can be effectively nullified by changing the weights and biases of the first layer. The first forward-propagation calculation is a linear combination after all. Normalising inputs of an ANN is common practice nonetheless. It turns out that having a uniform scale among the inputs, for instance $x \in [0, 1]$, helps in accurately determining the optimal step direction. When viewing a step in an N-dimensional euclidean space, the step is the euclidean distance determined from combining the different optimisation variables. If one component is significantly larger or smaller than others when using a non-normalised data set, it may happen that it dominates the optimisation process and prevents proper convergence [14]. Ultimately, it is good practice to employ normalisation in order to diminish biases towards select dimensions in a data set.

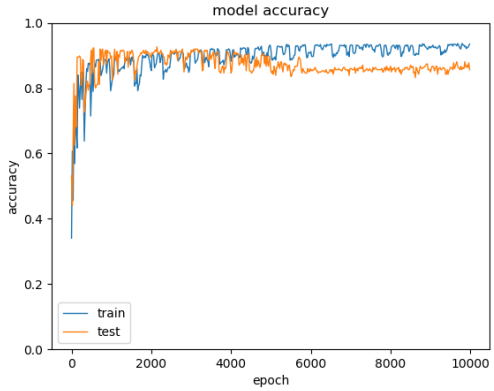


Figure 2.10: Training and testing accuracy during an ANN optimisation

Optimisation parameters The learning rate and decay are chosen based on the progression of training and testing errors throughout the training process. A visual example of such an error progression is shown in Figure 2.10. Based on such a plot, one can tune hyperparameters for the optimisation process according to the following guidelines.

- If training and testing accuracy are low, but both follow an increasing trend during the training process: increase training time for more accuracy and/or increase the learning rate;
- If the accuracy increases at first, but stagnates at some point in the training process: Decrease learning rate and/or decrease damping. This issue may also be caused by too few neurons and/or hidden layers;
- If large fluctuations are observed: Decrease learning rate and/or increase damping;
- If a high training accuracy & low testing accuracy is observed: Decrease training time. This issue may also be caused by many neurons and/or hidden layers.

Apart from the learning rate and decay, other optimisation parameters may be required such as β_1 and β_2 in the Adam optimisation algorithm. Such terms generally have a recommended value motivated by the developer of the algorithm, which facilitates a stable solution in most cases [22]. It may be beneficial to change these quantities, although this will mostly have to be explored through trial and error.

Initialisers The initial set of values assigned to the weight matrices and bias vectors in an ANN are of influence on the training process. One may never initialise these quantities with the same values on different entries, as this does not 'break the symmetry' [3]. The meaning of this statement is that when two or more values are the same in a weight matrix or bias vector, the derivative of the cost function with respect to these values are identical as well and thus, the

step according to the back-propagation algorithm is identical. Therefore, if these values are not randomly initialised, a network cannot learn properly.

Several popular initialisers exist that assign initial weights and biases randomly in a predefined range. This is done either in a uniform way or based on a normal distribution. All of these methods initialise weights centered around zero, but they vary in range and standard deviation respectively. While many initialisers exist, the following are the most popular ones. rand(x_1, x_2) indicates a random uniform distribution between x_1 and x_2 whereas N(μ, σ^2) generates a normal distribution with mean μ and variance σ^2. Furthermore, N_{in} and N_{out} are the amount of inputs and outputs of an ANN layer respectively.

- Random uniform and normal: rand$(-0.05, 0.05)$, N$(0, 0.05)$;
- Lecun uniform and normal: rand$(-\sqrt{\frac{3}{N_{in}}}, \sqrt{\frac{3}{N_{in}}})$, N$(0, \sqrt{\frac{1}{N_{in}}})$;
- He uniform and normal: rand$(-\sqrt{\frac{6}{N_{in}}}, \sqrt{\frac{6}{N_{in}}})$, N$(0, \sqrt{\frac{2}{N_{in}}})$;
- Glorot uniform and normal: rand$(-\sqrt{\frac{6}{N_{in}+N_{out}}}, \sqrt{\frac{6}{N_{in}+N_{out}}})$, N$(0, \sqrt{\frac{2}{N_{in}+N_{out}}})$.

To illustrate the difficulty in choosing an initialiser, the following statement is quoted from a well-known group of researchers and authors in the field of ANN. *Modern initialization strategies are simple and heuristic. Designing improved initialization strategies is a difficult task because neural network optimization is not yet well understood. [. . .] Our understanding of how the initial point affects generalization is especially primitive, offering little to no guidance for how to select the initial point.* (Goodfellow et al. [3]).

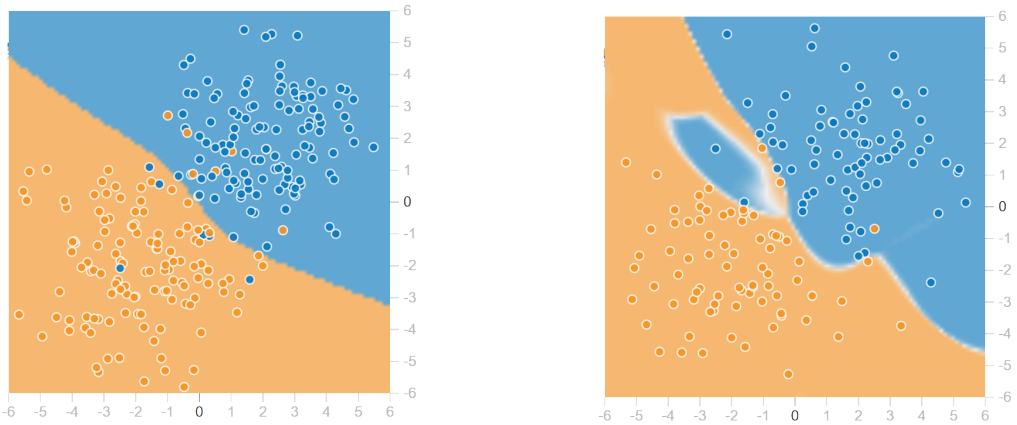
Under-fitting: Intuitively, this phenomenon is the opposite of over-fitting. Essentially, the issue manifests itself when an ANN is incapable of sufficiently capturing non-linear relations that are in the data. For example, a linear perceptron only classifies based on a straight line with a certain orientation. If the data is parabolic, this neuron is incapable of accurately classifying the data shape and will try to fit a straight line through the data by using, for example, least squares regression. If this happens, it is said that the ANN has a high bias [3].

A large error on both the training and test set indicates under-fitting. Such an error mainly arises due to short training times and/or too little complexity in the network. Under-fitting is easy to detect and means that the chosen model is not able to properly generalise relations from a data set. If this issue is encountered, the only solution is to modify hyper parameters (model settings). One should first check if a longer training time improves the model performance. If that doesn't help, it should be considered to add nodes and/or hidden layers. Another option is to look into other types of activation functions or optimise the learning parameters. The approach to solve under-fitting therefore depends largely on trial and error, as there are no robust methods to compute the optimal hyperparameter settings [14].

Over-fitting: While ANNs are famous for their capability to cope with noisy data, it may occur that training on too little data points results in a network that is fitted to noise as well. When this happens, the testing error is significantly larger than the training error. One of the possible solutions is to use more training data, if available, such that the noisy data samples do not have a prominent influence on the training process. Figure 2.11 visualises this issue: The correctly trained network in Figure 2.11a neglects the noise and consequently has a training error equal to the testing error, while Figure 2.11b shows how the noisy data is also captured. Consequently, the latter features a test set error which is 10 times higher than the training error which makes it a bad predictor. If a network over-fits on a data set, it is said that the ANN has a high variance [3].

The simplest way of solving over-fitting is to reduce the network complexity. Less nodes per layer and/or less layers means that the network is less capable of modelling details or outliers and therewith, less prone to the occurrence of variance.

A second method to reduce over-fitting is to implement an algorithm which stops the training process early [14]. A longer training time increases the chance of over-fitting the training data,



(a) (b)

Figure 2.11: Well-fitted (a) and over-fitted (b) ANN

so if it's possible to detect the ideal point to stop. The procedure is to split the available labeled data into three sets: training, development and testing. During the training phase, the training set is used to optimise the ANN and the development set is used to see how the 'development error' (similar to 'test error') progresses. As the training process continues, it may occur that the development error starts to increases significantly again which indicates the occurrence of over-fitting. At that point, the training is stopped and the weights are used at the training time which had a minimum development error. Finally, the test set is used as independent means of measuring the test error of the network.

Dropout regularisation is a third method which is able to reduce a high variance [5]. It's a simple procedure in which a random number of nodes are made inactive for a certain forward and backward propagation loop. This means that the weights to these nodes are set to zero, only for that loop. The result is that less nodes are trained simultaneously which reduces the complexity of adaptions to optimisation parameters and thus, reduces the likelihood of over-fitting.

A final means of preventing over-fitting is called (L1 or L2) regularisation. This method entails the addition of a penalty term to the cost function, which reacts to weight magnitudes. Fitting on single data points occurs by adding significant amounts of complexity (and thus, weights) to the system. The modified cost function therefore tends to reduce less significant weight values as much as possible, where the significance is measured by how many data points are labeled correctly due to a weight assignment. Intuitively, the consequence will be that the cost function tends to ignore outliers and the ANN complexity tends to reduce to the minimum required to model the general trends [14]. Obviously, this also implies that a too high amount of L1 or L2 regularisation may result in under-fitting.

2.3. Employment of Artificial Neural Networks in FEA: State of the Art

The field of modelling with ANNs has gained an explosive amount attention over the past decades. While the technique itself exists since 1940, it was not until the 90's that other engineering fields started to investigate the usability of this universal approximator into their own area. The availability of large amounts of data and the ever-increasing usable processing power, even for private individuals, contributed to this increase in popularity.

As of today, engineering fields such as image and audio processing are not imaginable anymore without ANNs. This is not true for structural analysis, as the applications of what is thought to be

a 'black box' is not immediately intuitive in a field where accuracy and understanding of physical phenomena mean everything. still, the application of ANNs in structural analysis and particularly FEA has been investigated continuously over the past 30 years, although not in a voluminous way. Most of these works recognise the issue that accurate simulations of complex structures to compute the structural response and/or damage progression still require considerable amounts of computational time and power. As a global approximator, ANNs may offer a (partial) solution to this issue.

The current section will first covers a relatively straight-forward strategy in which desired global output quantities are directly modelled with an ANN. This versatile approach allows to replace an existing model in its totality and is presented in Subsection 2.3.1 to show the capability of ANNs. Then, a more integrated approach is covered in Subsections 2.3.2 and 2.3.3 where the implementation of ANNs as constitutive model in FEM is investigated to approximate the elasto-plastic material response, as well as damage behaviour. Finally, several other promising data-driven techniques in the field of structural analysis are elaborated on in Subsection 2.3.4.

2.3.1. Direct Modelling of Problem Outputs

In general, it is agreed that FEA is a costly business from a computational point of view. Therefore, researchers have tried to speed up calculations by omitting the simulations completely for a specific application. The idea is to train an ANN on a data set that contains specific outputs of a simulation or experiment which are of interest for the research, such as critical buckling strength. Such an approach can be used, for example, to predict a critical buckling load or load at which damage is initiated. This section aims to provide an overview of the possibilities and limits of such ANN models by means of existing literature.

FEA-based optimisation A multi-variable optimisation process may easily require more than a thousand cost-function evaluations, according to Papalambros and Wilde [17]. If the cost-function is an analytical expression which takes less than one-hundredth of a second to evaluate, the total optimisation time is acceptable. This may not be the case for an optimisation problem where the cost function evaluation includes a complicated non-linear FEM simulation of a composite panel.

The work of Bisagni and Lanzi [23] recognises the efforts made by other authors to perform a minimum weight optimisation of a stiffened composite panel. Boundary conditions and outputs often included minimum panel strength and the critical buckling load. Inputs include the layup definition as well as stiffener count and dimensions, which means both discrete and continuous variables are involved. To cope with such variables as well as a potential non-convex cost function, a genetic algorithm (GA) is chosen as optimisation algorithm. As this approach is computationally expensive, Bisagni and Lanzi propose an optimisation procedure in which the desired outputs of the structural response are predicted by means of a system of ANNs as shown in Figure 2.12. These networks are first trained on a series of FEM simulations, after which the calculation can be omitted in the optimisation process itself. A separate ANN is trained for each output variable: Pre-buckling stiffness, buckling load, collapse load and even the load-displacement curve. Each ANN featured a relatively simple architecture with only one HL and a limited number of nodes. The only ANN with more than 10 nodes is the load-displacement curve predictor which is intuitive due to the various inputs such a curve depends on. The networks consistently predicted the desired results with more than 75% of training and test data within a 4% error margin. The improvement in computational time of this approach in comparison with an FEA-only optimisation is around an order of magnitude. A comparable approach is taken in Ruijter et al. [24], where FEA output data of a stiffened panel with two holes is used in a GA weight minimisation algorithm. The ANN structure used here features one HL and a limited number of nodes, similar to the work of Bisagni and Lanzi.

In Lanzi et al. [26], the same method is used on a more complicated problem: An explicit dynamic FEA of an impact event. The high complexity of this problem translates to increasingly complex non-linear relations between outputs and inputs. Both this work and Bisagni and Lanzi choose to use a system of (simple) ANNs, each representing one specific output variable. The

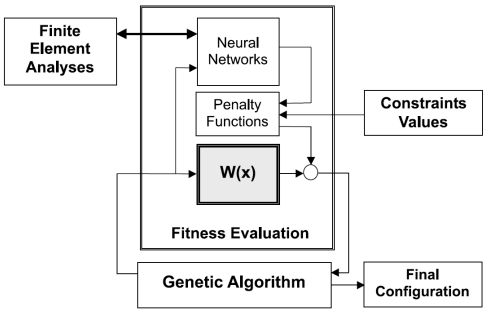


Figure 2.12: Optimisation procedure which employs trained ANNs as substitution for FE analyses [23]

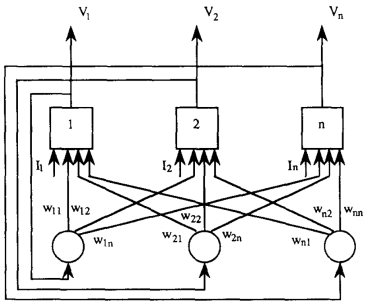


Figure 2.13: Example of a Hopfields Network architecture [25]

reason is to minimise the data set dimensions for each ANN, which results in efficient training. The work of Lanzi et al. applies the approach to a riveted (aluminium) tube as well as a honeycomb structure. For the riveted tube, the ANN is able to capture the trends in the load-time curve quite well with most predictions being within 1% and a few outliers up to 11%. For the honeycomb structure, the crush strength is modelled where the ANN succeeded to capture the global strength as a function of time with errors between 0 and 10%. Finally, a complex FE crash analysis model of a helicopter floor segment consisting of several frames is used as training object with similar results. This shows that a relatively simple ANN can indeed be used to predict results such as critical stresses based on a FEM model, despite the complexities.

Another approach in the same category is to predict the deflection of beam and/or truss elements under various types of loading. The deflection of a simple truss structure at each node is modelled in Hajela and Berke [25], with weight (proportional to geometrical stiffness) as input. With constraints on the maximum deflection, the weight of the structure is minimised. In contrast with previously named works, some unconventional ANN set-ups were employed. Both an Adaptive Resonance Network and a Hopfield Network were shown to be able to accurately model the truss problem with small errors. These ANNs are, however, less efficient in training and execution as they are not feedforward networks (i.e. they contain feedback loops, as seen from Figure 2.13).

In a recent work, Bessa and Pellegrino [27] explore the value of integrated data-driven computational procedures in optimisation. Herein, the optimal design of ultra-thin carbon fibre deployable shells under biaxial bending is considered. The significant load carrying capacity in the post-buckling regime, after the initial bifurcation point, leads to the need for an efficient means of obtaining the initial and ultimate buckling points. A data-driven framework is developed which is able to accurately predict the initial bifurcation point based on an output data set of pre-computed detailed simulations. Furthermore, it can construct the probability distribution of the ultimate buckling points which are heavily influenced by potential imperfections. The data-driven framework has been shown to operate both effectively and accurately for the composite shell optimisation problem.

Prediction of experimental results An engineer often desires to make predictions based on experimental data instead of numerical results. It has been shown in various works, such as Lanzi et al. [26], that significant differences may exist between experimental and numerical results despite the availability of a detailed FEM model. Fu et al. [28] demonstrates this approach by predicting the punch radius of a multi-step air-bending forming process as a function of springback radius, sheet thickness and yield strength divided by material stiffness. A small ANN with 1 hidden layer with 7 nodes was chosen in order to minimise the chance of over-fitting. The trained network achieved relative errors of less than 1.5% on all test data points. In addition, Fu et al. and Shahani et al. [29] constructed an FE model which turned out to be less accurate than the ANN predictor and take significantly more time when used in an optimisation cycle.

Another beneficial advantage of ANNs is their ability to deal with noisy data. In Levin and

Lieven [30], this was demonstrated in the dynamic analysis of an FE beam, where an ANN is used to update the model based on experimental data. The results show that this updated model is able to dramatically improve the error between experiments and numerical simulations, even with noisy training data.

2.3.2. Constitutive Modelling of Homogeneous Materials

When setting up an FE analysis, the engineer is required to specify one or multiple constitutive models which define each element's behaviour. Its core function is to relate the stresses and strains in an element to each other, such that it represents the mechanical response of a chosen material. In practice, this implies that not only the elastic material behaviour but also more complicated relations such as the plastic behaviour, brittle fracture and strain-rate dependencies may have to be captured in a mathematical model. The complete material behaviour is generally derived from extensive amounts of experimental data according to four steps, as motivated by Ghaboussi et al. [31].

1. Gather experimental data of the general material behaviour;
2. Construct a mathematical model which captures the basic observed behaviour;
3. Extrapolate by predicting non-trivial stress-strain states outside the experimental domain on which the model is initially based and compare the results with specific, suitable experimental results;
4. Modify or extend the model to account for the observed complex behaviour.

While this traditional approach of material modelling has been widely used since the time of Robert Hooke [31], alternative approaches have been proposed. One particular method involves the use of ANNs, in which their self-learning property is employed to directly build a (partial) material model from experimental or numerical material data. When considering an FE solution strategy, such as the one shown in Figure 2.14, the ANN is then most often implemented at the material level. A requirement for properly training such a network is the availability of sufficient data, which captures all aspects of material behaviour that are of interest. The first successful attempts to represent a constitutive model by a trained ANN date back to around 1990. The research of Ghaboussi et al. [31] represents an example of such an early endeavour as it investigates an ANN trained on the behaviour of biaxially loaded concrete.
The current subsection elaborate on the work done on this topic for homogeneous material models, whereas the specific possibilities of modelling laminated composite materials will be covered in Subsection 2.3.3.

Elasto-plastic behaviour One example of an ANN constitutive model is one which takes the current stress and strain state of an elasto-plastic, 2-D plane strain shell as inputs, as well as the stress increments along both axes. It then predicts the strain increments as output after passing the inputs through two hidden layers, each composed of 40 nodes as seen in Figure 2.15. This ANN is trained on a diverse range of load cases, including biaxial tension, biaxial compression and mixed tension/compression. It is proven that the network was able to generalise and predict material behaviour for arbitrary load cases while keeping the requested stress-strain states and stress increments in the training domain. In addition, it is investigated how the network responded to input data outside this domain. It turned out that these predictions are less accurate, but still reasonable in the sense that the general curve shape is captured despite an error in predicted stress up to 15%.

Apart from the static biaxial load case with constant loading ratios, Ghaboussi et al. [31] considers cyclic loading behaviour which is a complex non-linear phenomenon. This complexity comes from the fact that the stress-strain state after an increment depends not only on the current situation, but also historic states. It was determined by trial and error that two historic points on the stress-strain curve were adequate for the purpose of a uniaxial cyclic loading prediction,

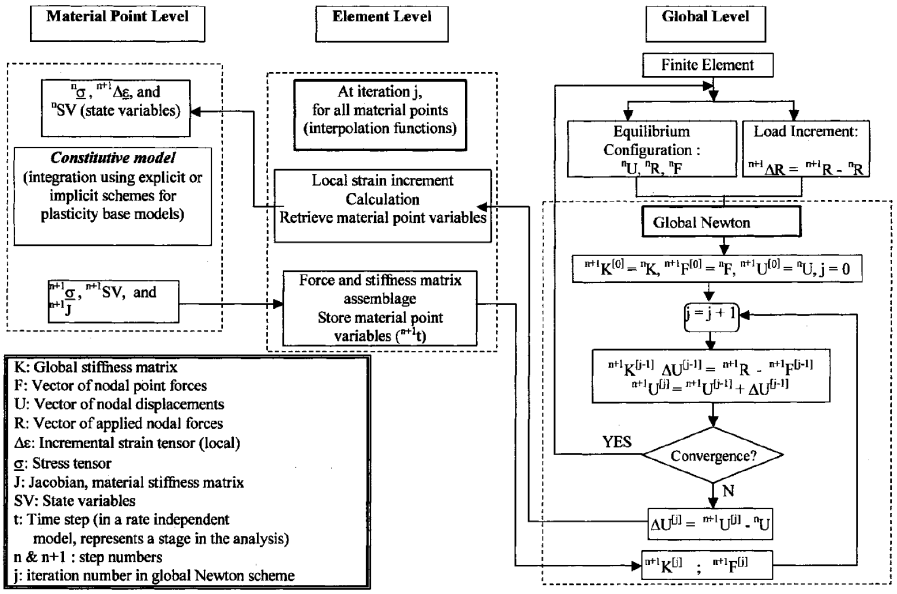


Figure 2.14: FEA solution strategy using a Newton-Rhapson scheme [32]

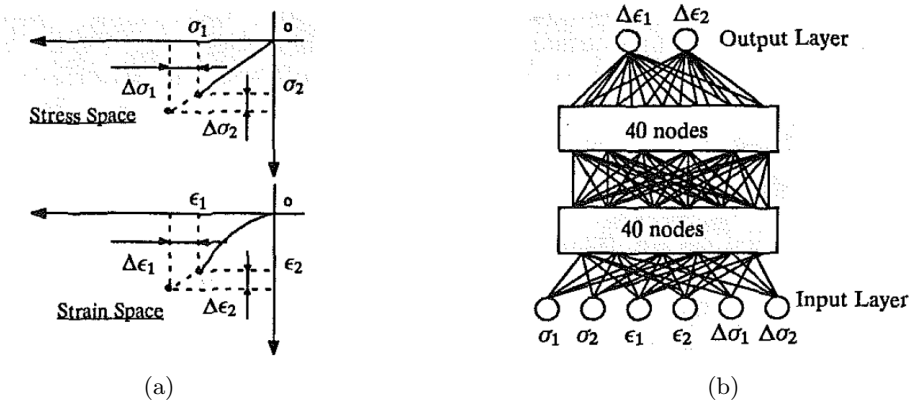
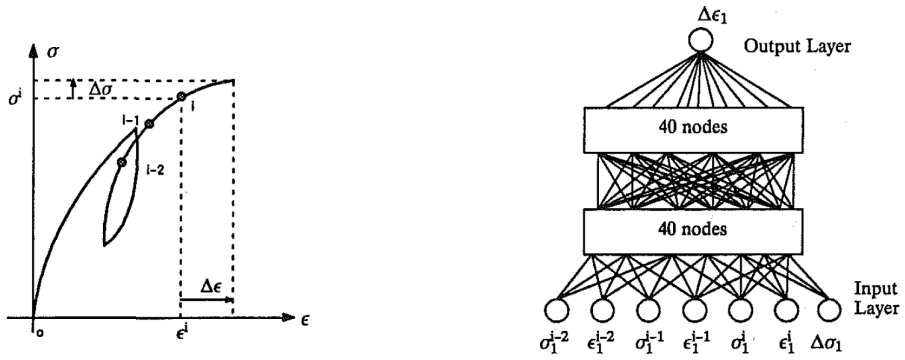


Figure 2.15: Problem variable space (a) and neural network architecture (b) as presented in Ghaboussi et al. [31]

resulting in an ANN architecture as shown in Figure 2.16. The network was shown to capture the cyclic behaviour within the training data region well and was even found to be able to generalise. In other words, one could train an ANN on data from four cycles and predict how the fifth and sixth cycle are shaped. The results of such an extrapolating approach show room for improvement however, when the cycles that are not part of the training data differ in their shapes.

Multiple demonstrations of ANN constitutive models are presented in Javadi et al. [33]. Similar to the work of Ghaboussi et al., the ANN is trained to predict the stress-strain behaviour based on the current state and a given increment size. Javadi et al. implemented a trained ANN in each element in an FE model as a substitution for the conventional constitutive material model. Apart from the computational procedure in each element, the analyses are performed by a conventional FE solver. The approach is tested on a 2-D beam with 10 elements and a transverse end-load, where the ANN is shown to be as accurate as the regular FE analysis. Furthermore, a quarter segment of a 2-D thick cylinder shell with nine 8-node elements subject to a distributed load is investigated. Again, an excellent agreement with both theoretical and FEM results is achieved with the ANN constitutive model. Finally, an analysis is performed involving the more complex



(a) (b)

Figure 2.16: Problem variable space (a) and adapted network architecture (b) to capture cyclic behaviour, as presented in Ghaboussi et al. [31]

constitutive behaviour of an embankment consisting of a non-linear Mohr-Coulomb soil material. The ANN, trained on triaxial experimental tests results, is shown to be potentially more robust than an FEA as it is based directly on the experiments instead of a mathematical model that is subject to assumptions.

ANN architecture A challenge with the previously mentioned approaches remains in choosing the ANN hyperparameters, such as the amount of HLs and nodes per layer. These decisions are often based on the engineer's experience and trial & error as there are no robust rules to determine hyperparameters [14]. The work of Ghaboussi et al. [34] elaborates on a method to improve the constitutive ANN modelling approach with an autoprogressive training algorithm. This type of algorithm modifies the network architecture during the training process based on convergence criteria, as shown in Figure 2.17. A practical implementation is performed on a truss structure subjected to transverse loading, where an ANN is trained to predict the constitutive behaviour for each element. The network initiation is done by pre-training on a small region of the elastic material behaviour instead of the conventional random weights assignment, which improves the training time. It then successfully adapts its dimensions during training based on the required modelling complexity.

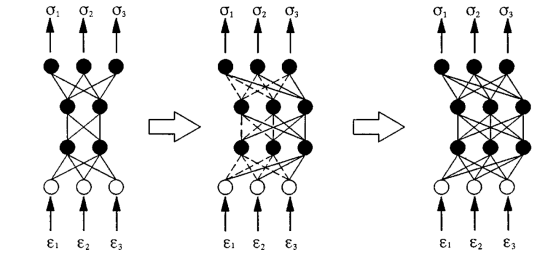


Figure 2.17: Autoprogressive training algorithm [34]

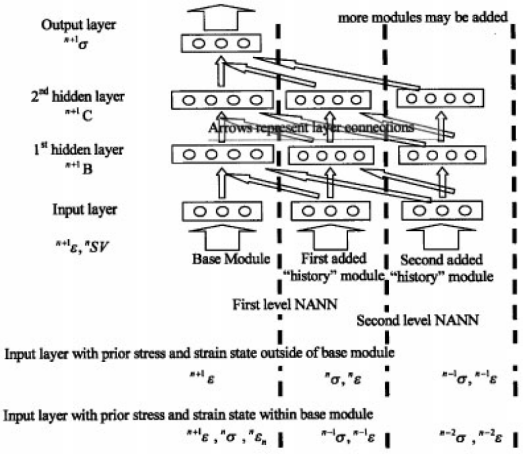


Figure 2.18: Encapsulated ANN architecture [32]

The effectiveness of the autoprogressive training principle is confirmed by Hashash et al. [32], where a more advanced version of this architecture including history input modules is employed.

This Nested Adaptive Neural Network (NANN) was originally proposed by Ghaboussi and Sidarta [35] and features a multi-layer architecture with predefined connections, similar to the one shown in Figure 2.18. The amount of nodes in each layer is flexible and may change depending on the training progression while preserving the predefined connections between 'nested' groups of nodes. By including historic stress-strain states in the input vector, this architecture is suitable for modelling the influence of the load history (such as cyclic loading or strain-rate dependencies).

Stiffness matrix calculation The ANN constitutive models described above act at the material point level: they compute a new stress-strain state based on a load increment and the current state. When reviewing a solution strategy for an FEA such as the one in Figure 2.14, Hashash et al. [32] recognises that one major issue lies in the difficulty of computing the material stiffness matrix in an explicit way. ANN constitutive models generally omit the stiffness matrix when calculating the element response, which is a limitation when considering that these matrices are an input to the element level of a traditional FEA. The work of Hashash et al. derives a relation which allows to explicitly compute the material stiffness matrix from the already present ANN definition. Thereby, an ANN constitutive model is defined that encloses the full material point level shown in Figure 2.14. The approach is verified by comparing the stiffness matrices of a simple square element during loading with the computed matrices from the ANN, where excellent agreement is found. Furthermore, a validation is performed on a cantilever beam subject to a concentrated end load and an excavation wall made of a highly non-linear synthetic soil. Both case studies show good residual force convergence characteristics and accuracy of results.

In a follow-up work, Hashash proposes the inverse material modelling approach shown in Figure 2.19 without the constraint of having a pre-defined constitutive model [36]. This is done by using an element-level ANN. Conventional models are an idealised and generalised approximation of material behaviour, based on a limited amount of experiments. A self-learning simulation is established that employs an ANN, trained on complementary boundary measurements of forces and displacements. The usage of both force and displacement measurements is an attempt to train a more accurate ANN model. Therefore, stresses from the load controlled analysis and strains from the displacement controlled analysis at material point level are used to from stress-strain pairs that ultimately form the training and testing data sets. The autoprogressive algorithm presented by Ghaboussi et al. [34] is employed to train and shape the network on experimental data on metal specimens. A boundary value problem, namely a plate with a hole under tension within a similar stress-strain range is then solved using the ANN within each element. The behaviour of the regular and ANN model are found to be in excellent agreement. Stresses and strains outside the trained range are tested as well, from which it is found that the ANN is not capable of extrapolating properly.

A specific focus on the inverse modelling of cyclic behaviour (reloading patterns) exists in the work of Yun et al. [37, 38]. The approach is similar to Hashash et al. [36], as both force and displacement controlled FE analyses are employed to train the ANN accurately with an autoprogressive training algorithm. The results are verified with both FE and experimental analyses of an axially loaded 3-D tube, as well as a complex transversely loaded frame structure. The ANN was able to predict the hysteresis behaviour of the studied objects with good accuracy.

Damage Capturing the initiation and progression of damage is often a crucial part of constitutive models. One possible approach to approximate the damage state in a material would be to use ANNs as a general damage predictor based on strains in a structure. These strains can be measured, for example, by piezo-electric actuators as proposed in Teboub and Hajela [39] or strain gauges on the surface as proposed in Kudva et al. [40].

The work of Kudva et al. states that damage deduction may be achieved through clever use of strain sensor measurements at discrete locations. Indeed, it is demonstrated for an aluminium-stiffened panel that an ANN can be trained on an FEA data set to predict a damage size and location with strain measurements as input. The amount of training examples is equal to the amount of predefined possible damage locations times a predefined amount of damage sizes. This

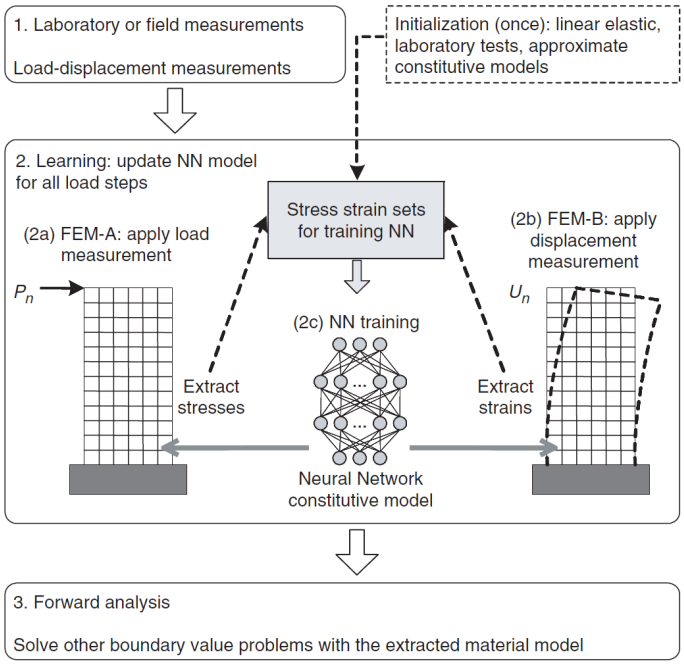


Figure 2.19: SelfSim ANN analysis framework [36]

results in a large amount of required training data and thus, a high computational cost which is also found in the work of Teboub and Hajela [39], as described in Subsection 2.3.3. The chosen network architecture in Kudva et al. [40] featured two hidden layers, as seen in Figure 2.20. While this is a higher degree complexity compared to the work of Teboub and Hajela [39], it should be noted that the way of measuring the strain field compensates for the complexity as only surface strain was measured with strain gauges. The method presented in this paper achieves a good accuracy on predicting damage locations, but features erroneous results in damage size predictions. This is attributed to the damage location being taken as discrete variable in this specific application in contrast to the continuous damage size variable.

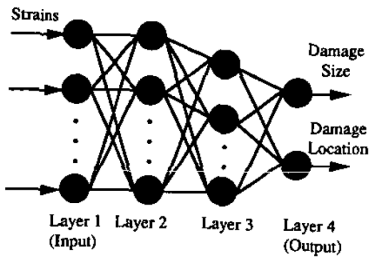


Figure 2.20: Layout of an ANN which predicts damage size and location from a given strain field [40]

2.3.3. Constitutive Modelling of Composite Materials and Laminates

In the aerospace industry, one particularly interesting class of materials are laminated composites. While accurate modelling approaches for these materials are available nowadays, the computational cost remains an issue. The extensive amount of complex non-linear phenomena in composite materials requires an equally extensive constitutive model, which is why researchers try to find alternative solutions.

Elasto-plastic behaviour of plies A step towards such a solution is proposed by Pidaparti and Palakal [41], who built a ANN model of graphite-epoxy laminae under several orientation angles. The network is trained directly on experimental data and predicts the non-linear stress-strain curves of plies under tension, as well as cyclic loading behaviour. The work of Ghaboussi et al. [34] proposes a similar approach, which involves the macro-scale modelling of a composite plate with an open hole. A ANN is used as a constitutive model and is trained directly on experimental data. The fundamental material is considered to be a single unidirectional ply, following the macro-scale approach. The Classical Laminate Theory is then used to assemble the laminate and construct the ANN inputs and outputs in every ply as visualised in Figure 2.21. The inputs of the network consist of the 2-D strain components of a ply (ϵ_1, ϵ_2, γ_{12}) and produces the stresses as outputs (σ_1, σ_2, τ_{12}). An autoprogressive algorithm is employed for training and the network parameters are pre-trained on elastic material data, as already elaborated on in Subsection 2.3.2. The agreement with experimental results is considered reasonable by the author, but potential for improvement remains. This is attributed to the absence of path-dependence in relation to the material response in the ANN. A nested modular network is proposed as a solution, which includes historical data as inputs. Furthermore, only in-plane loading and damage modes can be represented in a ply material model which can be improved by training the ANN on 3-D data.

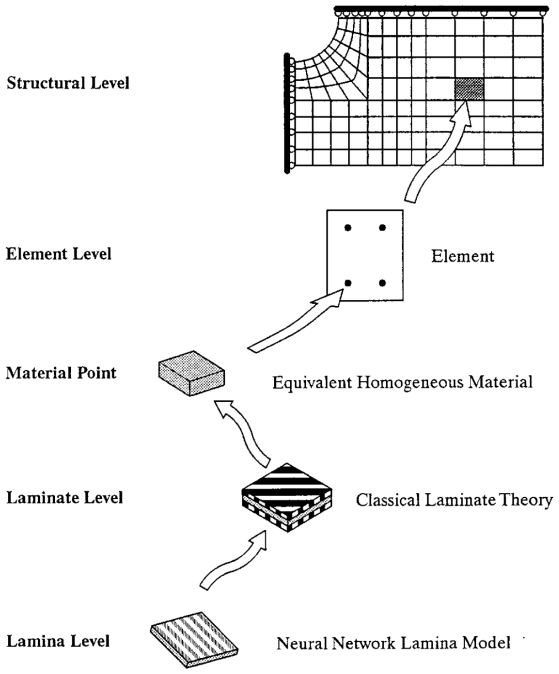
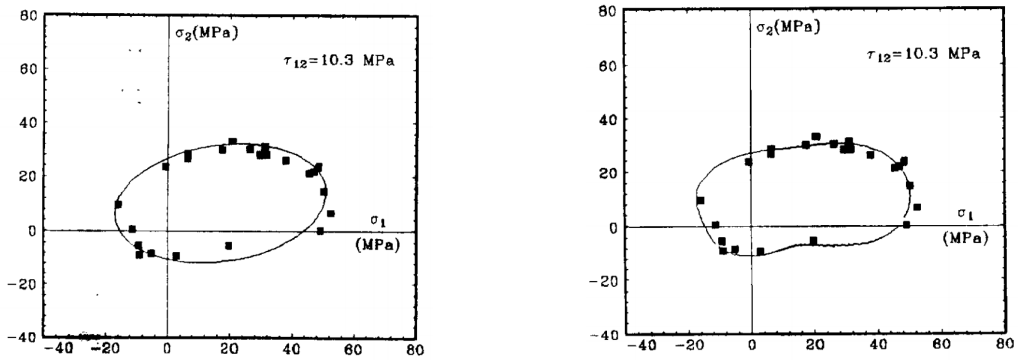


Figure 2.21: FE modelling approach for laminated composites with an ANN lamina model [34]

Strength The strategy presented in Labossiere and Turkkan [42] is an example of general failure prediction of thin, orthotropic laminae under bi-axial loading and plane-stress conditions. A comparison is presented of the analytical failure envelope and an envelope constructed by an ANN that is trained on experimental data, based on the biaxial ratio. While the former method approximates this failure envelope as an ellipse, it turns out that the ANN is able to represent a more accurate predictor. The possibility to construct an arbitrarily shaped failure region turns out to be beneficial, as seen in Figure 2.22. The disadvantage of this approach is found to be the

lack of physical significance, as the ANN architecture does not involve physical phenomena but solely relies on data instead.



(a) (b)

Figure 2.22: Comparison of failure envelopes of a biaxially loaded lamina approximated by an optimised tensor polynomial theory (a) and a trained ANN (b) [42]

In Lee et al. [43], biaxial strength predictions are performed by an ANN, trained on experimental data from cross-ply carbon/epoxy tubes. Similar to the work of Labossiere and Turkkan, a comparison is made between an analytical method (optimised tensor polynomial method and Tsai-Wu) and the ANN. Several biaxial ratios were tested in both compression and tension, resulting in a total of 43 labeled data points to train the network. A relatively small architecture of two hidden layers and five neurons per layer is used. The resulting predictor is more accurate than the analytical relations, as shown by Figure 2.23. The training error did not achieve the desired convergence however, which is attributed to the set of training data possibly being too small. Improvements may have been achieved by using a different network architecture or training algorithm as well, which is not attempted.

An optimisation problem where an ANN is found to be a suitable replacement for FEA is found in the work of Lopes et al. [44]. The aim is to demonstrate an improvement in computational efficiency when performing a reliability optimisation of laminated composite structures. The desired output for each structure is the failure probability distribution as a function of mean applied load. Several cases are considered; a simple two-layer laminate under uniaxial tension, biaxial tension and general in-plane (axial, shear and moments) loading as well as a semi-cylindrical composite shell under a pressure load. The ANN takes material allowables, laminate configuration and loading conditions as input and provides the reliability curve as output. Two types of networks (Multilayer Perceptron Network and Radial Basis Network) are investigated, the difference being in the processing of neuron inputs type of activation functions. Both networks functioned well and were able to improve the processing time with 87% and 80% respectively, when compared to a conventional reliability analysis with only FEA. The accuracy did not decrease compared to the conventional approach.

Damage Teboub and Hajela [39] found ANNs with up to 4 layers layers capable of predicting fibre fracture, delaminations and matrix cracks respectively of a laminated composite plate. This plate features small encapsulated piezo-actuators. The method is numerically verified on a detailed model of a composite cantilever beam. The chosen ANN outputs for delaminations are the centre location and size of the delamination, where a prediction accuracy of about 98% is achieved. The matrix cracks are characterised by crack density and centre within a ply with an accuracy of approximately 99 %. Fibre breakage is modelled in a similar way, featuring the crack density and centroid of the damaged ply as network outputs and achieving an accuracy is 98%. A major point of concern with this approach is the computational cost, however, as the stated accuracies are only achieved after 30,000 (matrix cracks), 60,000 (delaminations) and even 200,000 (fibre breakage)

training cycles (epochs).

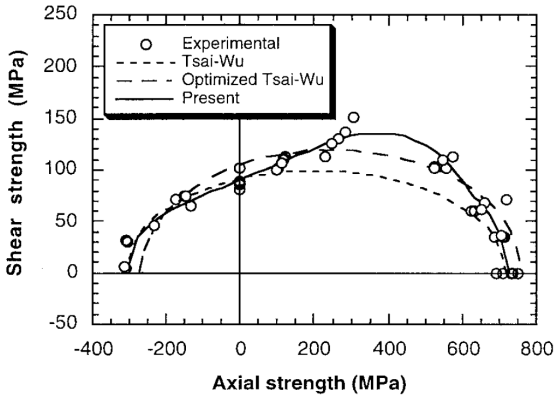


Figure 2.23: Comparison of failure envelopes approximated by analytical relations versus an ANN based on experimental data [43]

Another approach is presented in Man and Prusty [45], where a mathematical derivation is performed that allows for an energy-based damage modelling approach with an ANN. The method employs full-field strain measurements as well as boundary forces and displacements to train the network. It employs the principle of minimum energy to establish an error (cost) function. The effectivity of this approach is exhibited by modelling in-plane shear damage in a laminate based on experimental training data. A good resistance to noise and therefore the ability to generalise was demonstrated by the ANN. Especially interesting is the experimental validation, where a single uniaxial tensile test served as the basis for predicting multi-axial laminate behaviour.

2.3.4. Other Data-Driven FE Modelling Techniques

Apart from ANN constitutive models, there exist other strategies which smartly employ data directly into a numerical model. These so-called 'data-driven' techniques aim to improve the efficiency and/or accuracy of numerical computational procedures.

A well-written overview to data-driven mechanics is presented by Kirchdoerfer and Ortiz [46]. Herein, an alternative to traditional (empirical) material law modelling is introduced and presented. Instead, experimental material data, constraints and conservation laws are directly used to find an equilibrium material response. This approach directly employs material test data and finds the material equilibrium based on an energy minimisation formulation. It is shown that the data-driven approach converges to the classical approach when the latter is based on a high-fidelity data-set. It is proven that currently unmodelled materials or materials with highly complex behaviour may be modelled directly by the presented method.

The works of Nguyen and Keip [47] and Chinesta et al. [48] present a data-driven approach to non-linear elasticity and its integration in FEM by building on the research of Kirchdoerfer and Ortiz. A derivation is presented that leads to a direct formulation of the tangent stiffness matrix and force vectors as a function of data points. A validation is performed that showed good convergence characteristics and good accuracy, given that sufficient data points are available.

Data-driven methods can be useful to model composites as well. Liu et al. [49] developed a methodology that is capable of analysing a reduced representative unit cell (RUC), instead of a fully meshed (high-fidelity) version. Firstly, a set of full-scale computations are performed on a high-fidelity RUC. Then, several 'clusters' are identified by locating groups of material with similar mechanical behaviour by evaluating the strain concentration tensors of each element. A domain decomposition is then performed and between each cluster, an interaction tensor is defined as the influence of stress in cluster i on the strain in cluster j. The gathered information is then applied to perform calculations with the generated reduced RUC model and can be validated against the high-fidelity RUC's. The results of both analyses are found to match well and improve when

the amount of clusters is increased. Intuitively, increasing the amount of clusters also implies an increase in computational time. Still, a decrease of computational time of at least an order of magnitude is proven to be achievable for 2-D simulations, while maintaining the original accuracy. The approach is demonstrated on 3-D models as well but converges much slower due to the higher complexity. Due to this, 3-D reduced RUC did not prove to be sufficiently accurate and efficient to offer advantages over the traditional direct numerical simulation with a full mesh.

Bessa et al. [50] followed up on the work of Liu et al. by presenting several numerical experiments performed on a 2-D composite material RUC, both as high-fidelity and reduced order model. This approach considers three main sample properties in constructing a data set from the tested RUC's, as shown in Figure 2.24. The homogenised material properties and external conditions are defined for each experiment and a database is constructed from direct numerical simulations and reduced order models. Then, two distinct machine learning approaches are evaluated with which the sensitivities or characteristics of the model can be extracted from the data: Kriging and an ANN. Both methods perform similarly in terms of approximation error of homogenised RVE stress (<10% error) and strain energy density (<0.5% error). However, as Kriging requires expensive inverse matrix calculations, it is determined that an ANN is more suitable when using large data sets. The main recommendation presented by the authors was to use the presented approach in an optimisation cycle. Once a machine learning predictor is obtained, a global optimum can be found within a reasonable amount of time regardless of the original model size.

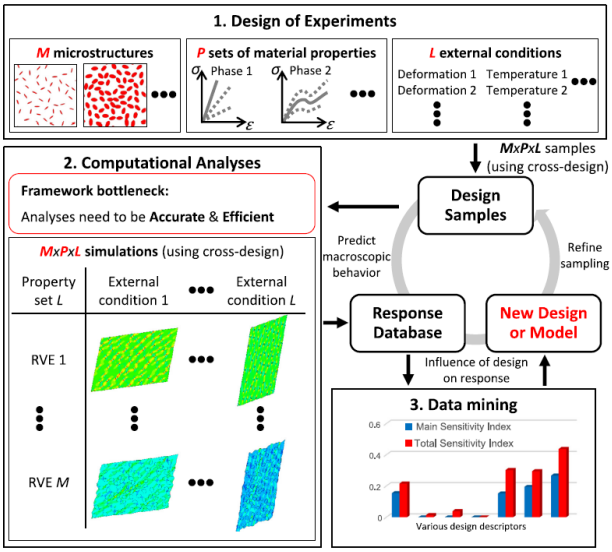


Figure 2.24: A data-driven numerical optimisation procedure, as presented by Bessa et al. [50]

A possible useful application in FEM of the above-mentioned data-driven material models is to employ so called Radial Numerically EXplicit Potentials (RNEXP). this method aims to form an approximation of a constitutive model based on sampling points, which contain information about the strain energy and loading state. RNEXP is essentially an interpolation scheme which omits the computation of field information, but rather focuses on the global (boundary) response. The sampling points may be obtained from experiments of numerical simulations. A drawback of RNEXP is the vast amount of data points required for an accurate simulation. In an attempt to solve this drawback, Fritzen and Kunc [51] recently published a two-stage approach that effectively employs RNEXP to perform simulations with high efficiency, without sacrificing accuracy. To generate the data efficiently, Fritzen and Kunc performed pre-computations with a Galerkin reduced order model after which the generated data is fed into the data-driven RNEXP simulation.

Apart from material modelling, another useful application of data-driven modelling is presented by Oishi and Yagawa [52]. This work proposes a method that employs machine learning to improve

a model's accuracy by estimating the optimal number of numerical quadrature integration points. A numerical optimisation is performed first to create a labeled data set, which is used to train ANNs with varying architectures. These networks are then used to predict the actual quadrature points. The error, however, turned out to be quite high with approximately 80% of the cases being predicted correctly for a 3-HL network with 150 nodes. Still, it is proven that employing the network to optimise the quadrature parameters, a more accurate solution is achieved compared to the standard Gauss-Legendre rule.

2.4. Summary

In the first two sections of this chapter, the goal was to establish a compact overview of what an Artificial Neural Network (ANN) is, how it functions and how to use it as a machine learning tool. The third section outlines the scientific work done concerning the application of data-driven techniques, with a focus on artificial neural networks, in the field of structural analysis and Finite Element (FE) modelling. The research questions regarding these topics, as presented at the beginning of the chapter, are answered below and summarise the most important findings.

Machine Learning with Artificial Neural Networks

To set up a machine learning problem, there are three main required 'ingredients'. First of all, the *task* is represented by the machine learning model. This model is essentially an equation whose shape depends on choices regarding its structure and properties. Secondly, the *performance measure* is the error function chosen by the user which evaluates the prediction error on a labeled data set and is used to train and test the model's performance. Thirdly, the *experience* is the collection of internal parameters which define the model behaviour at a certain point in time. The essential ingredients of a machine learning set-up with a neural network are therefore the network architecture and activation functions, the error or cost function, an initiation of internal parameters and not to forget; a labeled data set.

An ANN consists of multiple connected artificial neurons, also known as nodes. These nodes are simple processing units that apply a linear transformation to an input vector, followed by a non-linear activation function to produce an output vector. The weight matrix and bias vector of the linear combination in each neuron are optimised during the training process, where the goal is to choose the internal parameters such that the network can accurately estimate the output(s) corresponding to given input data. An ANN is a universal predictor: a network featuring one Hidden Layer (HL) with a variable amount of neurons can be trained to predict even the most non-linear relation, implying that there is no limit to a network's capabilities.

The training process of an ANN is based on a so-called 'labeled' data set, which consists of input data of which the corresponding result (output) is known. It can be generated through experiments, numerical simulations or another means. A data set is fit for training when it has sufficient data points, covers a domain that matches the application of the ANN and consists of all relevant inputs on which the output depends. The training process is an optimisation problem that minimises the cost function within a given domain. Thus, the network's optimisation parameters are adjusted by an optimisation algorithm. A popular method specifically suited for ANNs is the gradient-descent based backpropagation algorithm, which relies on the availability of an analytical derivative of the cost function with respect to each parameter to determine the optimal 'step' direction. This approach is only reliable for convex cost functions however, whereas non-convex problems require a more costly and complex alternative such as a genetic algorithm.

Bias and variance, also known as under-fitting and over-fitting, are commonly encountered issues that reduce the performance of an ANN. Bias indicates the lack of complexity caused by short training times or a network architecture that is too compact and can't capture non-linearities to a sufficient degree. It is resolved by lengthening the training time and/or increase the network size. Variance occurs when (noisy) data is too precisely fitted and the network is not generalising

properly. This issue is resolved by reducing the network complexity, reducing the training time and/or implementing a regularisation algorithm. The most common regularisation algorithms are dropout regularisation and L1/L2 regularisation. In the end, a good balance between training time, complexity and regularisation has to be found for a specific problem by trial and error.

Employment of Artificial Neural Networks in FEA

The first literature review question was to quantify to what extent an ANN is usable as a global substitute for a structural model. Various authors demonstrate the use of an ANN as a quick and relatively accurate predictor in an optimisation cycle. While it is not found to be quite as accurate as running a full FEM analysis for every cost function evaluation, this approach allows for a run-time reduction of multiple orders of magnitude. This way, a quick initial optimisation followed by a more precise analysis or an extensive global optimisation can be performed without facing the computational cost constraint. Typical modelled quantities include the critical buckling load, maximum deflection and critical stresses due to impact of a certain structure. The inputs consist of the same problem parameters as used during the regular analyses such as the material, structure and simulation descriptions. The usability and benefit of using ANNs as a global substitute for a structural model is therefore shown to be mainly related to quick cost function evaluations in an optimisation loop, when high accuracy is not of critical importance.

The second question was to find the possibilities and benefits of applying an ANN as a material model. It is observed that multiple authors investigated ANN representations of homogenised and laminated materials, in which the benefit lies in the absence of pre-defined equations and trends. An ANN is proven to be able to directly learn from experimental data and in many cases, provide a quick and more accurate way of establishing a material model compared to traditional methods. The demonstrated types of behaviour that can be modelled include the elasto-plastic behaviour, cyclic behaviour and strain rate dependent phenomena. The application of an ANN model to predict damage initiation and growth, as well as fatigue, is also part of material modelling and a possible application for ANNs. It was shown that an ANN can predict damage size and location in a homogenised material, with the strain field as an input. Furthermore, it was shown that damage modes such as fibre failure, matrix cracks and delaminations in laminae and laminates can be predicted with similar inputs. moreover, fatigue life of laminae and laminates can be predicted with good accuracy. It was found that the advantage of an ANN model is a higher accuracy compared to traditional methods through the independence of mathematical relations, which are subject to assumptions and boundary conditions. Ultimately, an improvement in computational time when using the ANN in a structural optimisation loop is shown to be potentially of multiple orders of magnitude. One limitation is the inability of an ANN to extrapolate and thus be used outside the training data domain.

In order to construct an accurate material model, the ANNs from literature are sometimes trained with an autoprogressive training algorithm. This type of training features an adaptive ANN architecture as there are currently no guidelines concerning the choice of hyperparameters (such as network dimensions). Furthermore, including historic data as ANN input was demonstrated to be an attractive approach to allow modelling of history-dependent phenomena. Network types that are most often used include the feed-forward network and radial basis function network, which differ mainly in the mathematical definition of the neurons. When using historic inputs, it is shown that an ANN architecture with encapsulated subnetworks and pre-defined connections can be especially useful for modelling material behaviour which depends on the load history. The network architecture and training times found in literature are relatively small concerning the prediction of load-displacement curves, in contrast to the case of damage prediction where larger ANNs and much larger training times were observed.

2.5. Discussion

In the current chapter, an attempt has been made to not only establish an overview of literature on ANNs and their application in structural analysis, but also to identify potential research topics.

All researchers agreed that a main goal of employing ANNs is the potential decrease in computational cost. Furthermore, the application of these self-learning functions may improve the accuracy of models where otherwise analytical relations are used, which are subject to assumptions and boundary conditions. It is intuitive from these advantages that ANNs may have a great impact when applied in FEA, which is a relatively costly method of structural analysis. Reducing computational time in FEA nowadays most often implies a reduction in accuracy as well. In search for a solution, researchers recognise the potential of introducing ANN-based material models that predict (non-linear) material behaviour, damage and fatigue. Such material models may be integrated in FEM directly as a substitute for the material level constitutive model.

While successful efforts have been made to demonstrate the potential of this methodology for both homogenised and laminated materials, there are some topics which still have potential. Based on the literature reviewed in this chapter, the following observations are made.

- ANN material models presented so far focus on either the elasto-plastic behaviour or the prediction of damage and failure. The combination of both has not yet been demonstrated in literature;
- The research to represent laminated composites by an ANN constitutive model has been focused on damage and failure predictions on ply level, whereas modelling of a laminate in FEM has not yet been demonstrated;
- Only few authors employ an ANN constitutive model which is integrated in FEM. While homogeneous materials are well-covered, laminated composites have only been investigated in 2D on ply-level.

From these observations, the research objective is be formulated. "The objective within the time-span of this thesis is to improve the accuracy and computational efficiency of a structural FE analysis by implementing a neural network constitutive model to be trained on a detailed and representative data set."

3

Constitutive Modelling Framework based on an Artificial Neural Network

For many years, engineers have sought to construct sets of homogenised properties to efficiently model complex material sections (e.g. domains with inclusions or cut-outs). Such a homogenised representation allows for an efficient means to capture the response of a section of a structure, from here on referred to as substructure, as if it is a homogeneous material domain.

When combining the idea behind a homogenised model with complex substructures, a main issue is how the homogenised material model should be defined mathematically. To capture the homogenised response of a substructure with non-trivial features, such as a plate with a cut-out, one requires a material model which is able to represent complex non-linear trends. As motivated in Section 2.5, an Artificial Neural Network (ANN) is a universal approximator that can be used to fill in this function. Based on a series of detailed Finite Element Method (FEM) simulations of a chosen substructure, an ANN can be trained to represent its homogenised behaviour. Subsequently, one could effectively construct an ANN-based constitutive model which can then be integrated into a FEM computational procedure. The main idea behind this ANN-based approach is that a homogenised constitutive model is able to represent the full complexity of a fully meshed substructure.

While an ANN is a powerful tool indeed, this work emphasises and recognises its major limitation; the ANN can only perform well if a properly defined and voluminous data set is available. The current chapter details the design and implementation of the proposed method as a framework for homogenised constitutive modelling with ANNs. As FE simulations are a rather expensive means of generating data, this work strives to construct the computational procedure in such a way that a minimum amount of simulations are required. In order to develop the ANN-based constitutive modelling framework, it is desired to facilitate quick iterations between substructures and ANN configurations. Therefore, this work only considers 2-D, implicit and static FE simulations. Despite this choice, the procedure will be applicable to any type of simulation with minimal modifications.

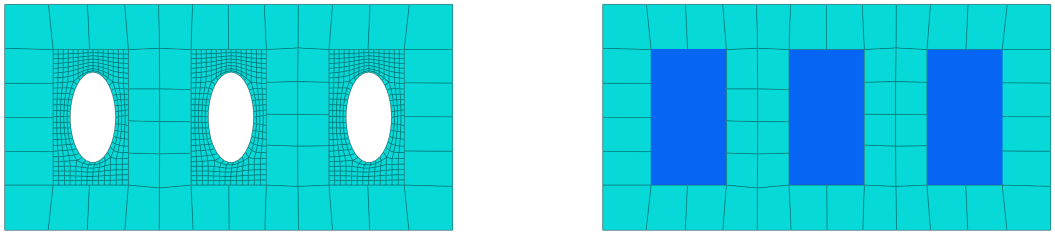
This chapter starts off by stating and motivating the main design decisions of the proposed computational framework Section 3.1. Based on these choices, a baseline ANN design is presented in Subsection 3.2.1, which will be the most simple in terms of inputs and outputs. Afterwards, three alternative ANN layouts which may lead to an improved performance are proposed in Subsections 3.2.2 to 3.2.4. A requirement to effectively integrate the ANN constitutive model in FEM is the tangential or secant material stiffness matrix calculation. The expressions for both these matrices are presented in Section 3.3. The programmed implementation of the ANN definition and training procedure, as well as the User Material Subroutine in Abaqus, are then discussed in Section 3.4. The chapter is finally concluded and summarised Section 3.5.

3.1. Designing the Computational Framework

Employing an ANN in structural analysis can be done in many different ways, but only in a handful of these cases is one actually able to take advantage of an ANN, without letting the disadvantages constrain usability. Section 2.3 outlines several approaches taken by researchers in the past, who all tried to find such an advantageous ANN application. Based on these studies, a research gap is pointed out in Section 2.5 which addresses the implementation of an ANN-based constitutive model in FEM. The current section builds on the identified opportunity and describes the design of a computational framework which incorporates the following functions.

- A constitutive model based on an ANN shall be created which is able to capture complex (homogenised) material behaviour;
- The ANN is to be trained on data capturing a chosen substructure's boundary response obtained from numerical simulations;
- The trained ANN shall be integrated in the FE software package Abaqus by creating a User Material Subroutine (UMAT).

To identify the benefits of this approach, it is illustrative to consider the 'substructuring' procedure commonly used in many engineering fields such as aerospace and automotive design. With this approach, most of the analyses and design actions are not performed on a complete structure. Rather, a domain decomposition is performed such that the structure is divided into two or more parts, each being subject to appropriate interface conditions. The design is then performed on each decomposed part at a fraction of the cost, after which the global structure is assembled again.



(a) (b)

Figure 3.1: Example of substructuring the densely meshed regions around each window (a) in a (simplified) fuselage panel and substituting them for one element with an ANN constitutive model (b)

The procedure proposed here is similar to the substructuring approach. When considering an FE model of a passenger aircraft fuselage, each window and its surrounding area can be viewed as a substructure. By decomposing a rectangular region around the window, a plate with a hole under varying loading conditions is obtained. If one would train an ANN to predict the mechanical response of this plate under each possible loading condition, it would be possible to substitute the network for each substructure occurrence. The complexity of the fuselage analysis is then reduced significantly as each window region can be represented by one element with a constitutive model based on the homogenised mechanical response of the substructure. This principle is visualised on a simplified fuselage panel in Figure 3.1.

With this explanation in mind, the potential benefits of the framework proposed in this thesis are well-defined. By substituting one or multiple partitions in a FE model by a single element with a trained ANN constitutive model, the following can be achieved.

- The proposed approach is effectively a mesh coarsening operation which, ideally, does not sacrifice accuracy and thereby reduces computational cost;
- As a limited amount of simulations are required to train the ANN constitutive model, an engineer may employ more detailed and accurate models for the substructure and obtain the

homogenised response. Once the ANN is trained on this, this improves the accuracy of all subsequent simulations where the ANN elements are employed, while maintaining the same computational cost.

A full scheme of the framework proposed in this thesis is visualised in Figure 3.2. The framework can be separated in three parts: the FE data generator, the ANN training module and the FE model with integrated ANN elements. Each of the following three subsections respectively elaborate on these parts.

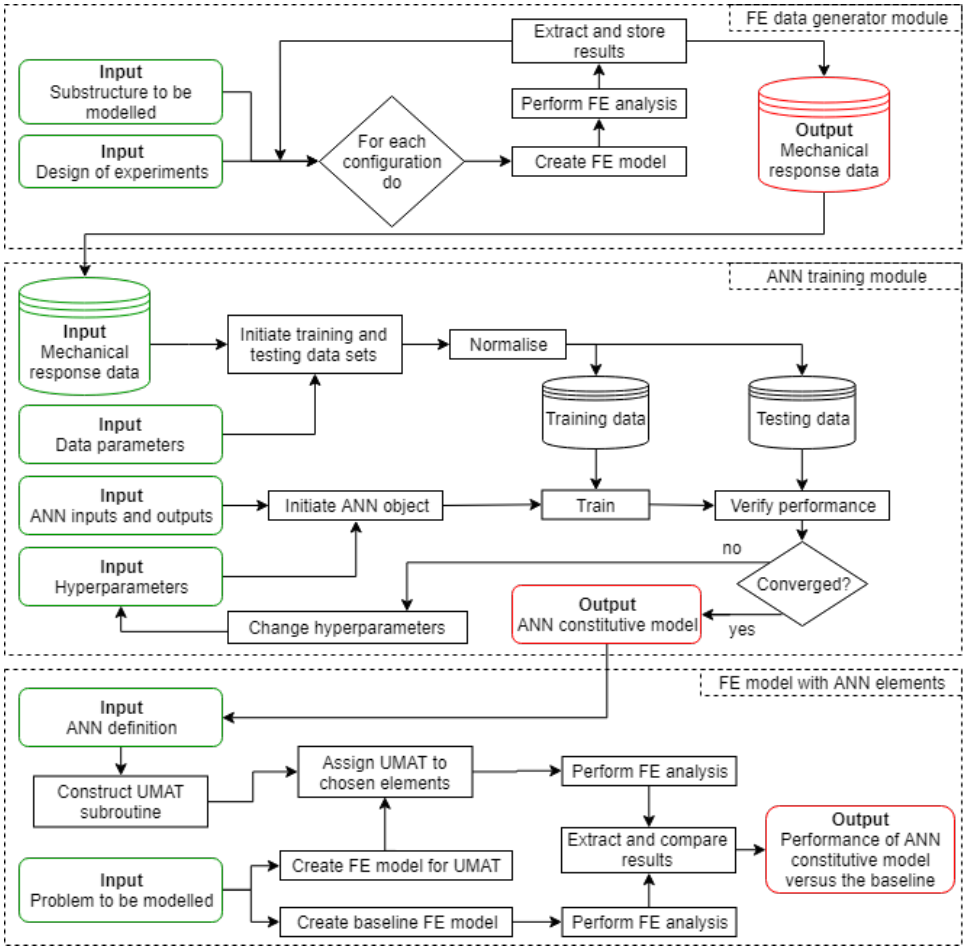


Figure 3.2: Schematic representation of the proposed computational framework

3.1.1. FE Data Generator Module

The purpose of the data generator module is to set up a representative set of FE models, perform the analyses and store the relevant results in a data set. It is designed to be fully automated by means of a script and perform the pre-processing, analysis and post-processing steps. While the full procedure of this module is extensively covered in Chapter 4, this subsection provides a compact overview of the inputs and outputs.

Substructure to be modelled First of all, one has to decide what substructure to apply the proposed framework to. In doing so, one should look for the following aspects.

- The substructure calculation causes a significant computational expense in the global model. The finely meshed region around each hole in Figure 3.1a is a good illustration of such a case;
- The forces and displacements at the boundary are uniform and thus, not of higher order. This allows the application of uniform strain-loading, which is required to set up a constitutive model. This is a desired property as it facilitates the mesh coarsening operation to take place, as shown in Figure 3.1b. When the size of the substructure is small compared to the global structure and only far-field loading is applied, this assumption holds;
- It is desired to choose a model on which periodic boundary conditions may be applied. It has been shown in literature, for example by Bessa et al. [50] and Chen et al. [53], that these boundary conditions are particularly useful for homogenisation, even when the structure considered is not fully periodic.
- (optional) The substructure repeatedly appears in the model in roughly the same form. This enables one to use the same constitutive model on multiple instances and improve the overall efficiency of the ANN-based approach.

In principle, any problem can be modelled if it complies with these properties. The demonstration models considered in the current work are a homogeneous steel plate and a composite plate with a hole described in Section 4.1. From the stated requirements, it can be reasoned that the homogeneous steel plate is not a good application due to the simplicity and low computational expense. In contrast, the composite plate with cut-out and damage phenomena will be a much better candidate to demonstrate the benefits of the proposed framework.

Design of experiments In general, the amount of simulations one needs to perform is closely related to the amount of independent variables the ANN takes as inputs. To properly generalise on the trends in a data set and correctly predict the network's outputs, the generated data set needs to comply with a few important points. Let us consider a hypothetical ANN with inputs $[x_1, x_2]$ and output y.

- A range for each problem variable needs to be specified. In most cases, it is already known what a reasonable range of the input and output variables is. For instance, it may be so that $x_1 \in [-100, -50]$, $x_2 \in [0, 1]$ and $y \in [-10, 10]$. This is defined as the domain and it indicates the range in which one data is to be obtained.
- For the considered application, the ANN outputs should only depend on the ANN inputs. In other words, y can be determined from x_1 and x_2 exclusively within the problem domain;
- A sufficient variation of each ANN input parameter needs to be present in the data set. Therefore multiple combinations of x_1 and x_2 shall be analysed in the data generator, for instance: [-100,0], [-100,0.5], [-100,1], [-75,0], ... , [-50,1]. The collection of these input combinations is called the simulation grid.

The current work deals with structural simulations. The input variables are the loads in each axial as well as shear direction, effectively resulting in 3 unique input variables for the FE Data generator when considering a planar 2-D model in a Cartesian $x - y$ coordinate system: ϵ_{xx}, ϵ_{yy} and ϵ_{xy}. For a 3-D case, there are 6 unique problem variables by the addition of ϵ_{zz}, ϵ_{xz} and ϵ_{yz}. With the guidelines stated above in mind, it is intuitive that the 3-D case requires many more simulations than the 2-D case due to the requirements on variation. In general, one chooses the amount of configurations within the domain to be analysed for each variable. In the example of x_1 and x_2, this number is 3 and it can be seen that the amount of data points to be evaluated is equal to $3^2 = 9$. If one more input variable would be added, however, 27 evaluations are required.

This exponential increase means that it is essential to evaluate which substructure, subject to how many variables, should be picked. 3-D structures already appear less attractive than 2-D structures and additionally, one should carefully evaluate which variables are desired for the modelled problem. The region with the hole shown in Figure 3.1a can be modelled in 2-D, with

the necessary variables being ϵ_{xx}, ϵ_{yy} and ϵ_{xy}. Additionally, one could also consider the size of the hole as an input variable. While this exponentially increases the amount of required simulations, it facilitates quick FE simulations with different hole sizes when employing the elements with an ANN constitutive model. This may be desirable when performing an optimisation, for example.

Mechanical response data The output of the FE data generator module is a data set, extracted from the performed FE simulations. After each analysis, the post-processing procedure described in Subsection 4.1.4 is executed. Based on that procedure, the data set for configuration looks as shown in Table 3.1. In this data set, the strains ϵ belong to the network inputs and the stresses σ are called the labels and form the outputs.

Table 3.1: Example data set resulting from one 2-D FE simulation

t_{inc}/t_{end}	ϵ_x	ϵ_y	ϵ_{xy}	σ_x	σ_y	σ_{xy}
0.00	-0.00	-0.00	-0.00	0.00	0.00	0.00
0.01	1.33e-4	2.00e-4	6.67e-5	10.4	14.6	6.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1.32e-2	1.98e-2	6.66e-3	118	230	-74.8

3.1.2. Artificial Neural Network Training Module

The ANN training module, as the name suggests, performs the optimisation or training process on a user-defined ANN. It requires the mechanical response data from the FE data generator module described in Subsection 3.1.1 along with several parameters to be chosen by the user. Furthermore, the ANN architecture is covered in more detail in Section 3.2. For details on the training procedure itself, the reader is referred to Subsection 2.2.4.

Data parameters While a data set containing all the information needed to train the ANN is already available from the FE data generator module, it still requires some formatting steps. In order to perform this formatting, the following data-related parameters are required to be set by the user.

- Train / test ratio: The current work employs a fixed division between training and testing data according to the 'holdout validation' principle. A fraction of data specified by the user is withheld from the training data set and used after training to check the ANN performance. The train / test ratio defines this fraction and normally ranges in between 80/20 to 95/5;
- Data set modifications: In order to explore different inputs and outputs for the ANN, it may be desirable to modify the data set. For example, the current work considers stress and strain increments, historic stress-strain states and a health parameter. These can be computed directly from the available data and are simply appended as additional columns.
- Normalisation parameters: Each ANN input and output is to be normalised before training as motivated in Subsection 2.2.5. For the current work, all quantities are mapped to the region [-1, 1]. The normalisation parameters are determined by identifying the minimum and maximum values for each input and output that can occur in the model.

ANN inputs and outputs The network is initiated with a certain combination of inputs and outputs. These can be chosen freely by the user, as long as the ANN outputs solely depend on the provided inputs for the chosen application. As the network may perform better when the inputs are presented in a different way, it may be desirable to try several options and compare the ANN performances. The following vectors are examples of possible inputs and outputs in the current work.

IN: $[\epsilon_x, \epsilon_y, \epsilon_{xy}, \sigma_x, \sigma_y, \sigma_{xy}, \Delta\epsilon_x, \Delta\epsilon_y, \Delta\epsilon_{xy}]$	**OUT:** $[\Delta\sigma_x, \Delta\sigma_y, \Delta\sigma_{xy}]$
IN: $[\epsilon_x, \epsilon_y, \epsilon_{xy}, \sigma_x, \sigma_y, \sigma_{xy}, \Delta\epsilon_x, \Delta\epsilon_y, \Delta\epsilon_{xy}, \zeta_x, \zeta_y, \zeta_{xy}]$	**OUT:** $[\Delta\sigma_x, \Delta\sigma_y, \Delta\sigma_{xy}]$
IN: $[\epsilon_x^t, \epsilon_y^t, \epsilon_{xy}^t, \sigma_x^t, \sigma_y^t, \sigma_{xy}^t, \epsilon_x^{t-1}, \epsilon_y^{t-1}, \epsilon_{xy}^{t-1}]$	**OUT:** $[\Delta\sigma_x^t, \Delta\sigma_y^t, \Delta\sigma_{xy}^t]$
IN: $[\epsilon_x, \epsilon_y, \epsilon_{xy}]$	**OUT:** $[\sigma_x, \sigma_y, \sigma_{xy}]$

The first case is the most simple case, where the ANN predicts a certain stress state given a strain state. The second example is an incremental approach: given the current stress and strain state and a strain increment, the ANN predicts the stress increment. Then, the third example considers not only the stress and strains at time t, but also the historic ones at time $t-1$. The motivation behind such an input vector may be to provide the ANN with a measure of the gradient of the mechanical response curve, which may help improve accuracy. Finally, the fourth example includes the health parameters ζ, which provide the network with a quantity that provides some extra information which it would otherwise have to deduct from the stress-strain state.

Hyperparameters This group of parameters includes all the global settings required to initiate the ANN, such as the network architecture and the optimisation settings. Subsection 2.2.5 provides an overview of which parameters belong in this group whereas the specific choices made for the current work are covered in Subsection 3.2.1.

ANN constitutive model This is the output of the ANN training module. The training procedure results in an ANN object that is optimised for the chosen data set. During training, only the weight matrices $\boldsymbol{w}$ and a bias vectors $\bar{b}$ are changed during training; the rest of the ANN definition is captured in the hyperparameters and ANN inputs/outputs.

3.1.3. FE model with Integrated Artificial Neural Network-based Elements

The final stage of the computational procedure is the integration of the ANN in a FE model. The ANN is implemented in a script that is usable by the FE software package Abaqus and in doing so, one can freely assign the ANN constitutive model to specific elements. This subsection covers the meaning of the inputs and outputs of the FE model with ANN elements shown in Figure 3.2.

ANN definition The output of the ANN training module is the definition of the ANN itself. Specifically, this includes the following information.

- Amount of inputs and output;
- Amount of hidden layers and nodes per layer;
- Activation functions used in each layer;
- A weight matrix $\boldsymbol{w}^l$ and a bias vector $\bar{b}^l$ for each layer.

The collection of this information enables the user to perform calculations with the ANN by a series of linear combinations and shape functions.

Problem to be modelled This is the actual application in which the ANN constitutive model is used. In general, the problem that is to be modelled will be the very starting point of the computational procedure. In other words, it does not specifically represent the substructure on which the ANN is trained but the global model instead.

Performance of the ANN constitutive model versus the high-fidelity model Let us consider the simplified panel with cutouts shown in Figure 3.1 as an example. An engineer may be dealing with such a model and thus, choose it as the problem to be modelled. Then, one or multiple areas may be identified to which the currently proposed framework can be applied.

In case of the panel, the densely meshed region around the hole would be a suitable candidate. The reason for this choice is that it is computationally expensive compared to the rest of the model due to stress concentrations and potential damage phenomena. The next step is to execute the FE data generator module and ANN training module to obtain the ANN constitutive model. Each hole with its surrounding region in the original model is now substituted for the ANN-based element, as is done in Figure 3.1b.

To gain an indication of the performance of this newly established model, a baseline FE analysis is performed on the conventional model. The performance of both FE setups can then be compared to provide the user with a quantification of how well the ANN model performs in terms of the mechanical response accuracy, as well as the computational time.

3.2. Designing the Artificial Neural Network

One of the main tools to be leveraged in the numerical framework developed for this thesis is the ANN. It is essentially an explicit equation in which the behaviour of a series of numerical experiments is sufficiently captured. It has been pointed out in Section 2.2 that many choices are to be made regarding the set up of this equation, as well as how it is optimised on the experimental data.

While Subsection 3.1.2 already provided an explanation of most required parameters in the context of the current work, the current section motivates a set of main choices in setting up the ANN. Four different ANN architectures and configurations are introduced, which are evaluated and compared in Chapter 5.

First of all, a baseline model is presented in Subsection 3.2.1. It contains the basic amount of inputs and outputs to properly function as a constitutive model, without derived quantities. Then, in an effort to facilitate a better approximation of the mechanical response after damage initiation, a set of health parameters are introduced as ANN inputs in Subsection 3.2.2. As a third option, a configuration with historic stress-strain states is presented in Subsection 3.2.3. Subsection 3.2.4 describes the final alternative: an ANN where the amount of inputs is reduced as much as possible, albeit accepting some limitations.

3.2.1. Baseline Design

To construct the ANN baseline model, it is essential to take a closer look at how the ANN is integrated in FEM. Subsection 3.4.2 discusses the implementation in further detail, from which a few key requirements can be deducted. First of all, the ANN shall be suitable for integration in an Abaqus UMAT subroutine and comply with the prescribed format [54]. This means that the inputs have to include the current stress-strain state. Furthermore, it is required that the ANN works with stress-strain increments in order to fit into an FE procedure.

The baseline ANN inputs and outputs are therefore the current stress-strain state. Furthermore, the strain increment vector is also an input while the output is the stress increment vector, corresponding to the UMAT subroutine output.

IN: $[\epsilon_x, \epsilon_y, \epsilon_{xy}, \sigma_x, \sigma_y, \sigma_{xy}, \Delta\epsilon_x, \Delta\epsilon_y, \Delta\epsilon_{xy}]$ **OUT:** $[\Delta\sigma_x, \Delta\sigma_y, \Delta\sigma_{xy}]$

Architecture While the input and output layer of the network are now fixed, the amount of hidden layers and nodes per layer remain to be chosen. This is an iterative procedure based on the guidelines stated in Subsection 2.2.5. A suitable architecture for each case in Chapter 5 is determined according to these guidelines. As an illustration, a baseline network layout with two hidden layers featuring N_1 and N_2 amount of nodes respectively is shown in Figure 3.3.

Activation functions These functions have to be chosen for each hidden layer and the output layer. Based on the review of the most commonly used activation functions provided in Subsec-

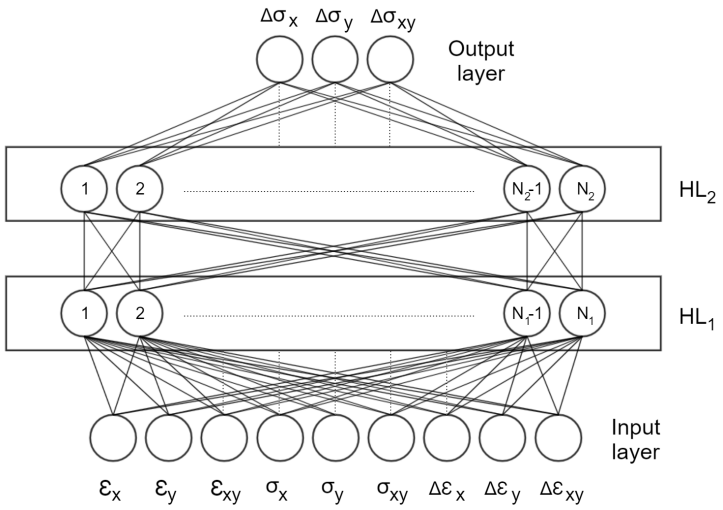


Figure 3.3: Baseline ANN architecture

tion 2.2.2, it is pointed out that the Rectified Linear Unit (ReLU) is the most suitable choice for each hidden layer. ReLU facilitates fast training and in a hidden layer, it is not required to use a function with a limited output range.

The output layer requires different considerations. It is pointed out that a limited range is indeed desired here, as the problem that is modelled has a limited range as well. Namely, the maximum values of the (normalised) incremental stress vector are already defined by the normalisation parameters, as explained in Subsection 3.1.2. Thus a function is chosen with a limited output range, leaving the sigmoid, hyperbolic tangent or SoftMax functions as possible candidates. As SoftMax is mainly intended for probability calculations, there is no benefit in using it for the current work. Furthermore the sigmoid and hyperbolic tangent functions have similar properties but in practice, the latter is the better performing choice [6]. Therefore, the hyperbolic tangent function is chosen for the output layer.

Cost function This function provides a measure of the error observed in the ANN output. As the derivative of the cost function with respect to each optimisation variable determines the direction of the optimisation step, this is a choice with significant influence on the training performance. The most popular cost functions which have proven their performance are outlined in Subsection 2.2.3. Since the currently considered case is logistic regression, qualified candidates are the Mean Squared, Mean Squared Logarithmic and Mean Absolute functions. It is impossible to choose the best among those without actually testing their performance in a verification case. However, after performing many iterations in the analysis phase of this thesis it is found that the best function is the Mean Absolute function.

Training algorithm After the derivative of the cost function with respect to the optimisation variables is computed, an increment has to be determined with which each optimisation variable is changed. The collection of these increments is called the optimisation step, which has both a direction (indicating the relative change of the variables) and a size (indicating the absolute change in the determined direction). Many algorithms exist to determine the optimisation step, although they are almost all based on gradient-descent with backpropagation. For details on how each algorithm works, the reader is referred to one of many informative sources such as Towards Data Science [1] or the book of Goodfellow et al. [3].

Ultimately, the best class of training algorithms that employ backpropagation are found to be

[1] `https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradien`

gradient descent optimisers with a momentum term. More specifically, the Adam optimiser with an adaptive momentum term called *Nadam* is most often found to be a good, stable choice for regular feed-forward ANNs and is the algorithm of choice for this thesis. This choice is backed up by the comparison between several optimisers presented in Tables 5.2 and 5.5.

Initialiser While the research towards ANN initialisation functions is a field of research on itself, it is clear that an ANN needs to be initialised with random weights in order to 'break the symmetry' of the network. If a random initialisation is not used, the network cannot properly be trained as several weights will always have the same value [3]. The introduction of randomness in an ANN means that each execution of the training process gives different results, albeit to a limited extent.

The take-away from this is that the choice of initialiser is determined by trial and error. Several initialisers are tested with the 2-D composite plate model and their performances are compared and summarised in Tables 5.1 and 5.4. From this overview it is concluded that both the Lecun Uniform and Lecun Normal initialisers are the best choice, although none of the tested schemes results in a particular bad performance.

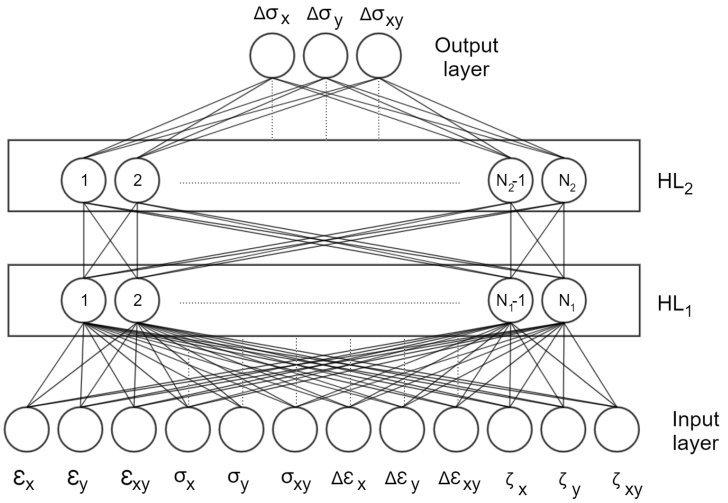


Figure 3.4: ANN architecture with health parameters as additional inputs

3.2.2. Including Health Parameters

The ANN with the health parameter as input is mostly identical to the baseline ANN presented in Subsection 3.2.1. While the baseline model contains all inputs that are required to determine the output, the currently proposed ANN accepts additional inputs that may 'help' the network to in achieving a higher accuracy or efficiency.

To illustrate this reasoning, let us consider the perspective of a hypothetical engineer who is highly experienced with the mechanical response of a certain material or structure. Such an engineer can, when shown a snapshot of the structure's mechanical response subject to a certain load, likely make quite accurate prediction of what will happen next. His prediction may be even more accurate if some additional information is provided, such as the current state of health in the structure. It is possible to apply this thought to the ANN and introduce health parameters ζ into the network. This indication of structural health is defined as the ratio of current secant structural stiffness (at time t) over the initial secant structural stiffness.

$$\zeta_{ij}^{t} = \frac{E_{ij}^{t}}{E_{ij}^{0}} = \frac{\sigma_{ij}^{t}}{\epsilon_{ij}^{t}} \frac{\epsilon_{ij}^{0}}{\sigma_{ij}^{0}} \tag{3.1}$$

With these quantities defined, the inputs and outputs for the 2-D ANN constitutive model with health parameters are as follows.

IN: $[\epsilon_x,\ \epsilon_y,\ \epsilon_{xy},\ \sigma_x,\ \sigma_y,\ \sigma_{xy},\ \Delta\epsilon_x,\ \Delta\epsilon_y,\ \Delta\epsilon_{xy},\ \zeta_x,\ \zeta_y,\ \zeta_{xy}]$ **OUT:** $[\Delta\sigma_x,\ \Delta\sigma_y,\ \Delta\sigma_{xy}]$

Apart from the different inputs, all other hyperparameters and choices regarding architecture, activation functions, cost function, training and initialisation of the ANN are equal to the ones chosen in Subsection 3.2.1. A visualisation of the ANN architecture including the health parameters is shown in Figure 3.4.

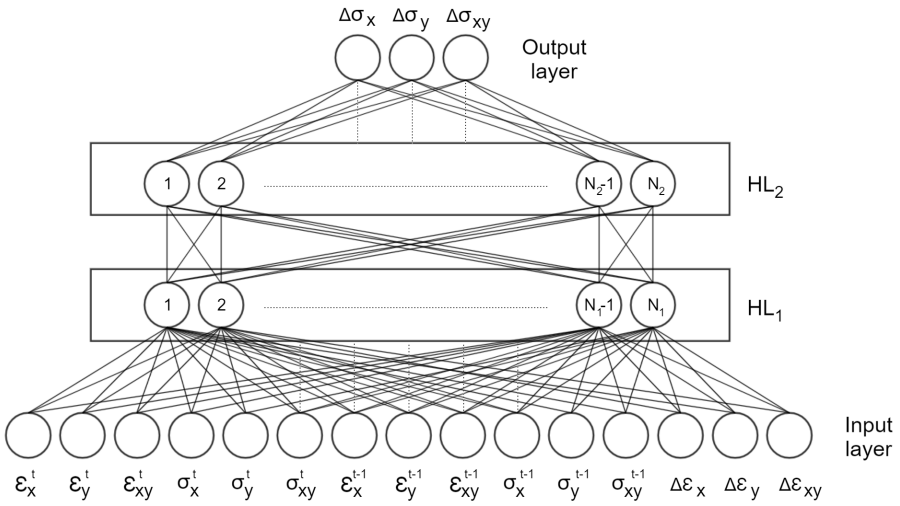


Figure 3.5: ANN architecture with historic stress-strain states as additional inputs

3.2.3. Including Historic Stress-Strain States

The ANN with the historic stress-strain states as input is mostly identical to the baseline ANN presented in Subsection 3.2.1. Similar to the ANN with health parameters as input, this network is fed additional inputs that may 'help' the ANN achieve a higher accuracy or efficiency.

The reasoning behind the inclusion of historic stress-strain states is that the network gains a means to regard the gradient of the mechanical response curves in its predictions. When considering the analogy of the experienced engineer used in Subsection 3.2.2, it is intuitive how this gradient may help determine the next step of the mechanical response. With these quantities added to the network, the inputs and outputs for the ANN constitutive model with the history module are as follows.

IN: $[\epsilon_x^t,\ \epsilon_y^t,\ \epsilon_{xy}^t,\ \sigma_x^t,\ \sigma_y^t,\ \sigma_{xy}^t,\ \Delta\epsilon_x^t,\ \Delta\epsilon_y^t,\ \Delta\epsilon_{xy}^t,\ \epsilon_x^{t-1},\ \epsilon_y^{t-1},\ \epsilon_{xy}^{t-1}]$ **OUT:** $[\Delta\sigma_x^t,\ \Delta\sigma_y^t,\ \Delta\sigma_{xy}^t]$

Apart from these changed inputs, all other hyperparameters and choices regarding architecture, activation functions, cost function, training and initialisation are identical to the Baseline case described in Subsection 3.2.1.

3.2.4. Minimum Input Design

The three previously proposed ANN structures feature inputs and outputs which correspond to those of the Abaqus User Material subroutine. It is, however, possible to further reduce the inputs to the following set, which is also visualised in Figure 3.6.

IN: $[\epsilon_x,\ \epsilon_y,\ \epsilon_{xy}]$ **OUT:** $[\sigma_x,\ \sigma_y,\ \sigma_{xy}]$

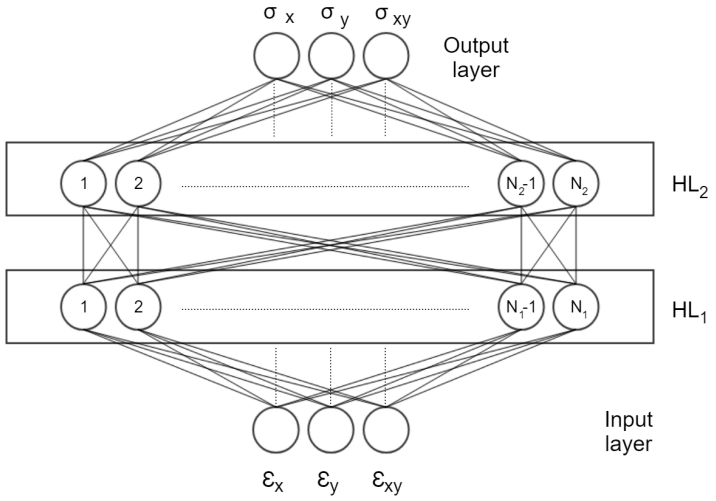


Figure 3.6: ANN architecture with a reduced input vector

It is important to recognise the advantages and disadvantages of choosing this input/output structure. The advantages are obviously the simplicity, as this network omits incremental stress-strain terms. This automatically implies that the size of the training data set can be reduced drastically, although the amount of required simulations remains the same. A disadvantage is that this network structure does not provide a sense of direction in the inputs, thus limiting its capability of capturing cyclic loading behaviour, for example. Furthermore it is not possible to compute the tangential material stiffness matrix directly which forces the user to employ the secant material stiffness, as is clear from the definitions and derivations covered in Section 3.3.

3.3. Computing the Material Jacobian

In a FE Analysis, a material's consistent jacobian matrix $\boldsymbol{C}$, also known as the incremental material stiffness matrix, represents the deformation of that material at the material integration point. Thus, it defines the constitutive behaviour at each iteration and may be defined as follows [54].

$$\boldsymbol{C} = \frac{\partial \Delta \bar{\sigma}}{\partial \Delta \bar{\epsilon}} \tag{3.2}$$

In this equation, $\Delta\bar{\sigma}$ is the incremental Cauchy stress vector and $\Delta\bar{\epsilon}$ is the incremental strain. It is valid for small-deformation problems or large deformation problems where the volume change is limited [54].

First of all, let us review why the material stiffness matrix actually exists. In traditional FE procedures such as the one shown in Figure 2.14, this matrix is computed by an explicit means to determine the incremental stress resulting from an incremental strain. For the FE constitutive model calculations, $\Delta\epsilon$ is the input and $\boldsymbol{C}$ represents the known material law. The multiplication of these two then results in the incremental stress $\Delta\sigma$, which is the output of the constitutive equations together with $\boldsymbol{C}$.

Another purpose of the stiffness matrix is found on the global level of the FE analysis. The global stiffness matrix is assembled from all element stiffness matrices. These are computed by evaluating $\boldsymbol{C}$ the consistent jacobian at each integration point in an element and performing a numerical integration procedure. The whole process is covered in-depth in the work of Zienkiewicz [55]. Ultimately, it is essential that the global stiffness matrix is accurately computed as the calculation of the displacement increment for each node is performed based on this matrix.

The integration of an ANN constitutive model in a FE procedure requires a different perspective

concerning the material jacobian calculation. It was already pointed out in Subsection 2.3.2 that ANN constitutive models generally omit the material stiffness matrix when calculating the element response. Instead of obtaining an explicit formulation for $\boldsymbol{C}$ from which the stress increment can be derived, the ANN constitutive model computes the incremental stresses directly. While this is an advantage on itself, it is also a disadvantage when considering that these matrices are required within implicit FEA procedures.To be able to integrate the ANN constitutive model effectively in FEM, the following subsections introduce several methods to compute the material tangent and secant stiffness matrices.

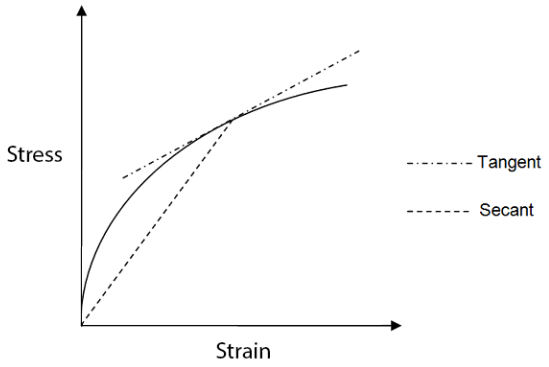


Figure 3.7: Visualisation of tangent versus secant stiffness in a mechanical response plot

3.3.1. Tangent Stiffness

The tangent stiffness is a matrix that defines the stiffness of a system when subject to a small incremental deformation. It contains the tangent stiffnesses of all combinations of stresses and strains, as shown in Figure 3.7. This quantity is used in the Full Newton-Rhapson and Modified Newton-Rhapson optimisation algorithm in FEM [55].

In order to perform a derivation, it is pointed out that the ANN constitutive model is essentially a function of the following form: $\mathrm{f}(\epsilon, \sigma, \Delta\epsilon) = \Delta\sigma$. Due to the explicit definition of an ANN, it can be evaluated as often as desired in an efficient way. The following paragraphs describe two approaches to compute the tangent stiffness matrix for a 2-D problem.

Systems of linear equations approach This is a straight-forward approach that applies small incremental strains in three directions and evaluates the ANN for each of these cases. As the stress-strain state is fixed, this provides three unique incremental stress and strain vectors. The conventional system of equations, relating the stresses to strains through the consistent jacobian, is defined as follows.

$$\begin{bmatrix} \Delta\sigma_x \\ \Delta\sigma_y \\ \Delta\sigma_{xy} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{bmatrix} \Delta\epsilon_x \\ \Delta\epsilon_y \\ \Delta\epsilon_{xy} \end{bmatrix} \tag{3.3}$$

By evaluating three unique increments, the vectors $\overline{\Delta\sigma}$ and $\overline{\Delta\epsilon}$ are both defined three times. This means that 9 unique equations are obtained corresponding to the 9 unknowns in the $\boldsymbol{C}$ matrix. Rewriting Equation (3.3) then results in three systems of equations; one for each row of $\boldsymbol{C}$. For $\Delta\sigma_x$, this system looks as follows.

$$\begin{bmatrix} \Delta\sigma_x^{(1)} \\ \Delta\sigma_x^{(2)} \\ \Delta\sigma_x^{(3)} \end{bmatrix} = \begin{bmatrix} \Delta\epsilon_x^{(1)} & \Delta\epsilon_y^{(1)} & \Delta\epsilon_{xy}^{(1)} \\ \Delta\epsilon_x^{(2)} & \Delta\epsilon_y^{(2)} & \Delta\epsilon_{xy}^{(2)} \\ \Delta\epsilon_x^{(3)} & \Delta\epsilon_y^{(3)} & \Delta\epsilon_{xy}^{(3)} \end{bmatrix} \begin{bmatrix} C_{11} \\ C_{12} \\ C_{13} \end{bmatrix} \tag{3.4}$$

While this method does the required job, it does not feature an optimal computational efficiency due to the triple evaluation of the ANN and system of equations solver.

Explicit equation derived from the ANN definition An efficient means of computing the incremental tangent stiffness matrix is covered by Hashash et al. [32]. This paper derived an explicit equation for the tangent stiffness by using the ANN architecture, activation functions and weights. As the paper presented the derivation only for the specific case where all activation functions are *tanh* and no biases are included, it is required to generalise it to apply in the current work. Here, the derivation is performed for an ANN with 2 hidden layers featuring *ReLU* activation functions, while the output layer features a *tanh* function.

The first step is to define the normalised input and output vector, indicated with $\bar{I}^{ANN}$ and $\overline{O}^{ANN}$ respectively during this derivation.

$$\begin{aligned} \bar{I}^{ANN} &= \frac{\bar{I}}{\overline{S}^{I}} \qquad \text{such that} \qquad -1 < \bar{I}^{ANN} < 1 \\ \overline{O}^{ANN} &= \frac{\overline{O}}{\overline{S}^{O}} \qquad \text{such that} \qquad -1 < \overline{O}^{ANN} < 1 \end{aligned} \tag{3.5}$$

Where $\overline{S}$ are the vectors with normalisation parameters for each variable. With this information, the values of nodes B_n in the first hidden layer can be determined as follows.

$$B_n = ReLU\left(\sum_{i=1}^{N_I} w_{ni}^{I1} I_i^{ANN} + b_i^{I1}\right) \tag{3.6}$$

Where N_I is the amount of nodes in layer I (input) and w_{ni}^{I1}, b_i^{I1} are the indicated entry of the weight matrix and bias vector between the input layer and HL 1. The second hidden layer nodes C_n and output layer nodes O_n are determined in a similar way.

$$\begin{aligned} C_n &= ReLU\left(\sum_{i=1}^{N_1} w_{ni}^{12} B_i + b_i^{12}\right) \\ O_n^{ANN} &= tanh\left(\sum_{i=1}^{N_2} w_{ni}^{2O} C_i + b_i^{2O}\right) \end{aligned} \tag{3.7}$$

Now that all nodal values are defined, Equation (3.2) is differentiated by using the chain rule to derive the expression for the tangential stiffness matrix entry i, j. From here on, the normalised strain and stress increments are referred to as $\Delta\epsilon^{ANN}$ and $\Delta\sigma^{ANN}$.

$$\frac{\partial \Delta\sigma_i}{\partial \Delta\epsilon_j} = \frac{\partial \Delta\sigma_i}{\partial \Delta\sigma_i^{ANN}} \frac{\partial \Delta\sigma_i^{ANN}}{\partial \Delta\epsilon_j^{ANN}} \frac{\partial \Delta\epsilon_j^{ANN}}{\partial \Delta\epsilon_j} \tag{3.8}$$

Recall the definition of the normalisation parameters and the equation can be rewritten as follows.

$$\frac{\partial \Delta\sigma_i}{\partial \Delta\sigma_i^{ANN}} \frac{\partial \Delta\sigma_i^{ANN}}{\partial \Delta\epsilon_j^{ANN}} \frac{\partial \Delta\epsilon_j^{ANN}}{\partial \Delta\epsilon_j^{ANN}} = \frac{\overline{S}_i^{O}}{\overline{S}_j^{I}} \frac{\partial \Delta\sigma_i^{ANN}}{\partial \Delta\epsilon_j^{ANN}}$$

Now let us differentiate the leftover partial derivative by parts by using the ANN layer definitions.

$$\frac{\partial \Delta\sigma_i^{ANN}}{\partial \Delta\epsilon_j^{ANN}} = \sum_{k=1}^{N_2} \left(\frac{\partial \Delta\sigma_i^{ANN}}{\partial C_k} \frac{\partial C_k}{\partial \Delta\epsilon_j^{ANN}} \right) = \sum_{k=1}^{N_2} \left(\frac{\partial \Delta\sigma_i^{ANN}}{\partial C_k} \left[\sum_{l=1}^{N_1} \frac{\partial C_k}{\partial B_l} \frac{\partial B_l}{\partial \Delta\epsilon_j^{ANN}} \right] \right)$$

To evaluate this expression, the derivatives of both activation functions are determined by the chain rule.

$$\begin{aligned} \frac{\partial \mathrm{ReLU}(f[x])}{\partial x} &= step(f[x]) \frac{\partial f(x)}{\partial x} \\ \frac{\partial \tanh(f[x])}{\partial x} &= (1 - (tanh[f(x)])^2) \frac{\partial f(x)}{\partial x} \end{aligned}$$

Applying this to Equations (3.6) and (3.7) leads to the following expressions.

$$\frac{\partial \Delta\sigma_i^{ANN}}{\partial C_k} = (1 - (\Delta\sigma_i^{ANN})^2) \cdot w_{ik}^{2O}$$
$$\frac{\partial C_k}{\partial B_l} = step(C_k) \cdot w_{kl}^{12}$$
$$\frac{\partial B_l}{\partial \Delta\epsilon_j^{ANN}} = step(B_l) \cdot w_{kl}^{I1}$$

At this point, Equation (3.8) can be worked out further and the following explicit expression for the tangential stiffness matrix is obtained.

$$C_{ij} = \frac{\partial \Delta\sigma_i}{\partial \Delta\epsilon_j} = \frac{\overline{S}_i^O}{\overline{S}_j^I} \sum_{k=1}^{N_2} \left(\left[\left(1 - \left(\Delta\sigma_i^{ANN} \right)^2 \right) \cdot w_{ik}^{2O} \right] \cdot \left[\sum_{l=1}^{N_1} \left(step(C_k) \cdot w_{kl}^{12} \right) \left(step(B_l) \cdot w_{kl}^{I1} \right) \right] \right) \tag{3.9}$$

3.3.2. Secant Stiffness

Compared to the derivations of the tangent stiffness matrix, the expression of the secant stiffness is straight-forward. As visualised in Figure 3.7, the secant stiffness is simply the instantaneous stress divided by the instantaneous strain. The components are thus formally defined as follows.

$$C_{ij} = \frac{\sigma_i}{\epsilon_j} \tag{3.10}$$

This expression can be employed with the Quasi-Newton optimisation algorithm in FEM. Despite being a much simpler and efficient expression that is not dependent on the architecture of the ANN, it is a good choice for the ANN constitutive model for the following reasons.

- The Quasi-Newton method performs better than the modified Newton-Rhapson algorithm. While it is not as quick to converge as the Full Newton Method, it is still a dependable choice which is especially efficient for solving a large system of equations [56];
- The secant stiffness matrix expression is independent of architecture;
- To compute the secant stiffness matrix, one requires only one ANN evaluation;
- Singularity issues are avoided as the determinant of the secant stiffness matrix is never ≤ 0.

3.4. Implementation

At this point, all required ingredients to define the ANN model have been discussed. Section 3.1 describes and motivates how the proposed computational framework is set up. The specific choices regarding the ANN module are presented in Subsections 3.1.2 and 3.2.1. Here, it is motivated how data parameters, inputs/outputs and hyperparameters are chosen. With these defined, it is time to implement the ANN model definition and training process, as well as the implementation in Abaqus.

This section covers the implementation of both the ANN model and its integration in Abaqus. When observing Figure 3.2, Subsection 3.4.1 describes the data processing, ANN initiation, training process and verification of the ANN performance. Then, the integration in Abaqus through a UMAT subroutine script is presented in Subsection 3.4.2.

3.4.1. Modelling Artificial Neural Networks in Python

In order to establish the ANN model, this subsection first chooses a machine learning library to utilise for an efficient implementation. Furthermore, the algorithm describing the various aspects of the ANN model is covered.

Choosing a machine learning library While ANNs can be programmed from scratch within a reasonable amount of time, numerous machine learning libraries exist in several programming languages that allow for a user-friendly implementation. While commercial parties such as Mathworks [57] and Wolfram Research [58] dominated the market for such software from around 1990 until 2010, the world of machine learning software is largely open-source today. In fact, due to the efforts of Google Brain Team [59] (*TensorFlow*), the Montreal Institute for Learning Algorithms [60] (*Theano*) and their respective developer communities, the open-source software is generally more advanced than commercial software. Apart from the preference to use an open-source library, the following aspects are weighed in the trade-off.

- **Language:** The preferred interface language is Python, as this is an accessible, general-purpose, object-oriented language with a large group of 'speakers';
- **Parallel execution:** In order to facilitate quick training of ANNs, this is a requirement. All popular open-source libraries support this, while commercial packages such as the *Matlab Deep Learning Toolbox* require additional (paid) add-ons and *Wolfram Mathematica* does not support it at all;
- **CUDA support:** This feature may increase the speed of ANN training and execution by performing calculations on the Graphics Processing Unit. While it is less beneficial for small ANNs, it proves a significant speed-up for larger ones;
- **User and developer community:** Not only the amount of users, but also the type of developers is essential to consider. Especially for an open-source library, it may be that bugs are present which remain either undiscovered or unresolved. A large user base helps discover bugs and a large (preferably professional) developer community ensures that improvements are made in a quick and efficient manner.

Based on these requirements, the choice is made to use *Keras* [61] (with *Tensorflow*), which is a powerful interface that is able to employ *Theano* or *Tensorflow* as back-end library. Both these libraries are excellent and comply with all above-mentioned requirements. By using the *Keras* front-end, the code may happen easily modified and expanded in follow-up works to suit these specific projects. Furthermore, these libraries are actively developed by professional programmers and even companies such as Google, which results in a practically bug-free code and an extensive amount of features. After performing the analyses presented in Chapter 5, the choice for *Keras* and *TensorFlow* is reinforced.

Scripted implementation The parameterised implementation of the ANN is a critical part of the proposed computational framework. Due to the utilisation of the machine learning library *Keras*, most of the code is, in fact, related to data processing. The script is set up such that it allows for quick iterations of ANN configurations and works for 1-D, 2-D and 3-D problems. An overview of the procedure within the script is outlined in Algorithm 1.

3.4.2. Abaqus UMAT Subroutine in Fortran

In order to construct a user-defined constitutive model in Abaqus, one needs to comply with a specific format [54, 62]. The subroutine has to be written in either *C* or *Fortran* and has the following fixed inputs and outputs. Here, t is defined as the current iteration and $t+1$ is the new

iteration.

$$\textbf{IN:} \quad \bar{\sigma}^t,\ \bar{\epsilon}^t,\ \frac{\partial \Delta\bar{\sigma}^t}{\partial \Delta\bar{\epsilon}^t},\ \mathrm{SV}^t,\ \Delta\bar{\epsilon}^{t+1}$$

$$\textbf{OUT:} \quad \sigma^{t+1},\ \frac{\partial \Delta\bar{\sigma}^{t+1}}{\partial \Delta\bar{\epsilon}^{t+1}},\ \mathrm{SV}^{t+1}$$

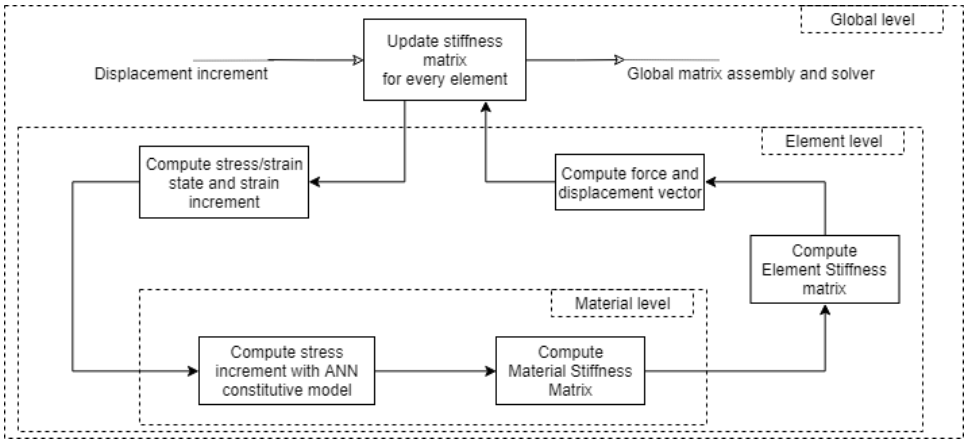


Figure 3.8: Illustration of how the UMAT subroutine fits into the FEA algorithm

The currently proposed UMAT will act on the material level of a FE analysis. To illustrate this, Figure 3.8 visualises the element and material level procedure of a Newton-Rhapson-based FE scheme. For every iteration and time increment, the FE procedure computes the global force and displacement vector for the whole model based on the incremental global stiffness matrix. This information is then split up per element and passed through the element and material-level calculations. The element subroutine first converts the forces and displacement increments to stresses, strains and strain increments at the integration points. Then, the material subroutine (ANN constitutive model) computes the updated stress state and material stiffness matrix. This information is passed back onto the element level where the element stiffness matrix and force vector are determined through numerical integration (Gauss quadrature) [55]. The procedure within one element is now complete and the computed force vector and element stiffness matrix is passed to the global level again.

The algorithm behind the ANN-based UMAT subroutine is shown in Algorithm 2, whereas the actual code can be found in the Github repository on `https://github.com/Dwergmaster/ABAQUS_ANN_constitutive_model`.

3.5. Summary

The current chapter proposed a constitutive modelling framework for the Finite Element Method (FEM) using Artificial Neural Networks (ANN) based on the research opportunity identified in Chapter 2. The goal was to develop a procedure that allows to capture the mechanical response on the boundaries of a substructure of choice in a single element. Its mechanical response was to be captured in an ANN definition, which was integrated in the FEM software package Abaqus through a User Material Subroutine.

The developed framework consists of three modules. Firstly, the FE data generator (extensively covered in Chapter 4) is a script that performs FE simulations on a chosen substructure under several loading combinations. The output from this module is a data set of mechanical responses at the boundary of the substructure. This data is fed into the second part of the framework: the

ANN training module. Here, the data is processed to a usable format. Furthermore, the ANN architecture is defined as well as the hyperparameters. After optimising the network, an ANN object is obtained that accurately predicts the structural response of the substructure. In the third and final module, this ANN is integrated in a UMAT subroutine in Abaqus and can be used as desired in a FE model. Here, it represents the substructure as one element and therewith, facilitates efficient and accurate simulations.

In Section 3.2, a more in-depth look was presented on the ANN model implementation. First of all, the choice of activation functions, cost function, initialisation and training algorithm were motivated. Then, four different ANN input and output configurations were proposed. The first one is a baseline model with the stress state, strain state and strain increment as inputs and stress increment as output corresponding to the inputs and outputs of a UMAT subroutine. Furthermore, two modifications with health parameters and historic stress-strain states were proposed which may lead to an improved capability of the ANN to capture the mechanical response. Finally, a 'reduced input' design was presented which only takes the strain state as an input and directly predicts the stress state.

Then, the calculation of the material stiffness matrix was considered in Section 3.3. Two derivations were presented to compute the tangent stiffness matrix; one explicit approach with a dependency on the ANN architecture and one based on the solution to a system of equations. While this proved the feasibility to directly determine the tangent stiffness, a more convenient and computationally efficient approach turned out to be the use of the secant stiffness matrix. This matrix is straight-forward and explicitly computable, independent of the ANN architecture and may be employed when the Quasi-Newton method is used.

Finally, the algorithms behind the scripted implementation of the ANN in Python and the UMAT subroutine in Fortran were detailed in Section 3.4. The accuracy of these scripts and the computational framework in general will be evaluated in Chapter 5.

Algorithm 1: Algorithm of the ANN training module

Input: Database with stress-strain data per increment, choice of verification case(s), amount of HLs, amount of nodes per HL N_l, activation functions ϕ, cost function, initialiser, optimiser, α, γ, β_1, β_2, normalisation parameters, N_{epochs}, N_{batch}

Result: weights $\boldsymbol{w}_l$ and biases $\bar{b}_l$, Plot of training and testing accuracy versus epochs, plot of verification case(s)

```
Initialise inputs;
for load_case in database do
    Import data;
    Normalise data;
    Sort columns in data set to match ANN inputs;
    if load_case in verification cases then
        Append data to testing data set
    else
        Append data to training data set
    end
end
Initialise sequential ANN object;
for l in layers do
    create new layer with N_l nodes and activation function φ_l;
    Initialise w_l, b̄_l
end
Compile the network with cost function and optimiser;
Train the network for N_epochs with N_batch data entries per epoch;
Plot accuracy progression of testing and training data;
for load_case in verification case(s) do
    Generate list of incremental loads from load_case;
    Normalise incremental loads;
    Initiate ANN reconstruction array with zero stress-strain and the first increments of
     Δε based on load_case;
    The reconstruction data array is the validation of a load_case and contains stress-strain
     data predicted by the ANN;
    for increment in increments do
        Perform ANN prediction of last increment in reconstruction array;
        Append prediction to reconstruction array
    end
    denormalise reconstruction array;
    Plot ANN mechanical response (reconstruction array) with validation data;
end
Request user_choice;
Based on the mechanical response, let the user choose if the fit is good enough;
if user_choice == "Yes" then
    Generate Fortran code for UMAT according to Algorithm 2;
end
```

Algorithm 2: Algorithm of UMAT subroutine

Input: N_{dim}, ANN definition, Normalisation parameters, $\bar{\sigma}^t$, $\bar{\epsilon}^t$, $\Delta\bar{\epsilon}^{t+1}$, $\frac{\partial\Delta\bar{\sigma}^t}{\partial\Delta\bar{\epsilon}^t}$, SV^t
Result: $\Delta\bar{\sigma}^{t+1}$, $\frac{\partial\Delta\bar{\sigma}^{t+1}}{\partial\Delta\bar{\epsilon}^{t+1}}$
Define type and dimension of all variables;
Construct ANN input layer vector from $\bar{\sigma}^t$, $\bar{\epsilon}^t$, $\Delta\bar{\epsilon}^{t+1}$ and SV^t;
Normalise input vector;
`Loop over hidden layers and output layer to compute ANN output;`
for *l in layers* **do**
  Allocate input $\bar{x}^l$, net input $\bar{z}^l$ and activation vector $\bar{o}^l$ dimensions;
  Assign local input vector $\bar{x}^l = \bar{o}^{l-1}$;
  Allocate weight matrix $\boldsymbol{w}^l$ and bias vector $\bar{b}^l$ dimensions;
  Assign weights, biases and input vector values;
  Compute $\bar{z}^l = \boldsymbol{w}^l\bar{x}^l + \bar{b}$;
  Compute $\bar{o}^l = \phi^l(\bar{z}^l)$;
  deallocate all layer-specific variables
end
`Output vector of final layer is the ANN output;`
$\Delta\bar{\sigma}^{t+1} = \mathrm{denormalise}(\bar{o}^{output})$;
Compute $\bar{\sigma}^{t+1} = \bar{\sigma}^t + \Delta\bar{\sigma}^{t+1}$;
Compute $\epsilon^{t+1} = \epsilon^t\Delta\epsilon^{t+1}$;
`Material stiffness matrix calculation (secant stiffness);`
Initialise stiffness matrix D with dimensions (N_{dim}, N_{dim});
for *i = 1,*N_{dim} **do**
  for *j = 1,*N_{dim} **do**
    D(i,j) = $\frac{\sigma_i^{t+1}}{\epsilon_j^{t+1}}$
  end
end
$\frac{\partial\Delta\bar{\sigma}^{t+1}}{\partial\Delta\bar{\epsilon}^{t+1}} = D$;
Update state variables SV^{t+1} `Optional`

4

Creating the Automated FE Data Generator

In this chapter, the Finite Element Method (FEM) models used to generate data for the machine learning process are discussed. More specifically, an overview is provided of how the models are built, the setup of the analysis itself and the post-processing procedure. The models are created in Abaqus/CAE and scripted in Python for automation purposes.

Firstly, two different FE models are designed and built in Section 4.1. Afterwards, the employed material models are elaborated on in Section 4.2. A particular feature of the models are the periodic boundary conditions, which are explained with a general example in Section 4.3. Then, Section 4.4 evaluates the performance and convergence of each model. Finally, a summary of the current chapter is provided in Section 4.5.

4.1. Model Creation

The current section elaborates on the set-up of the FEM models, which will be used as 'benchmark cases' in order to demonstrate and evaluate the framework proposed in Chapter 3.

In order to establish these models, Subsection 4.1.1 first outlines the sought-after characteristics within the scope of this work. After defining the problems to be modelled, Subsection 4.1.2 covers their actual set-up in FEM. The simulation settings are then provided in Subsection 4.1.3. Subsequently, the desired outputs from the post-processing phase and the methods to obtain these are detailed in Subsection 4.1.4. Finally, Subsection 4.1.5 provides a compact overview of the scripted implementation of the model creation, analysis and post-processing order to automate the procedure.

4.1.1. Choosing the Benchmark Cases

Since the current work aims to lay a baseline for the method introduced in Chapter 3, it is desired to choose a set of compact models as benchmark cases. The following factors are leading in determining the geometry and nature of these cases.

Shape As explained in Chapter 3, the procedure is to create a FE model of a substructure and train the ANN on the boundary response of this model under various loads. The shape of the model is therefore only constrained to feature a symmetry, such that it would be possible to create an infinite number of repetitions. Many shapes comply with this requirement; from simply shaped triangular prisms to a complex tetradecahedron [63, 64]. While the current work can be applied to all of these shapes, the benchmark is chosen to have a rectangular initial geometry. This results in

the most intuitive and straight-forward application of periodic boundary conditions and loading, and is therefore ideal to illustrate the approach.

Dimensionality Each Degree of Freedom (DoF) requires a number of inputs and outputs to the ANN. A 1-D problem will only have stress-strain quantities in 1 direction as an input, while a 2-D problem includes the stress and strain in the 2 axial directions including one additional transverse (shear) term. A 3-D problem features three axial and three transverse stress-strain quantities.

One major goal of this work is to demonstrate that a ANN is capable of capturing the inter-dependence of multiple loading directions. Therefore, a 1-D case is not an option, while the 2-D and 3-D case both comply with the modelling objectives.

Computational resources Another consideration in choosing the benchmark cases is the computational time to complete each simulation. To generate one data set for ANN training, anywhere between 25 and 100 FE analyses have to be created, executed and post-processed for a 2-D model. This number may increase to over 150 simulations for a 3-D model. This high amount of simulations is required in order to have enough variation in the input parameters such that the ANN can generalise properly. The computational resources available for the current work are limited and it is therefore preferable to consider 2-D cases.

Hyperparameters As explained in Subsection 2.2.4 and Chapter 3, an ANN based machine learning optimisation problem may comfortably require over 15 different hyperparameters to be set by the user. These parameters include the ANN structure, the activation functions, inputs and outputs, weight and bias initialisations as well as the optimisation algorithm and its settings. Although guidelines exist, the process of finding the optimal set of hyperparameters is generally considered to be highly iterative [6, 14].

A relatively efficient way of optimising the hyperparameters is to start with a simplified version of the model that is ultimately aimed at. The reduced computational time, but preservation of the type of inputs and outputs allows for a quick iterative process and results in a good initial guess for most of the hyperparameters. It speaks for itself that complexity-related quantities such as each hidden layer size and training time cannot be determined this way. However the optimal learning rate, decay, choice of activation functions, initialisation characteristics and even the amount of hidden layers can already be determined. From this simplified model, one can increase the complexity in steps until the full model is reached.

Table 4.1: Characteristics of the FE benchmark cases

#	Space	Material	Damage	Loading	Features
1	2-D	High-strength Steel (Elasto-plastic)	-	$\epsilon_x,\ \epsilon_y$	homogeneous
2	3-D	CFRP (QI)	Hashin	$\epsilon_x,\ \epsilon_y,\ \epsilon_{xy}$	elliptical hole

Taking the above-mentioned considerations into account, two different cases of increasing complexity are chosen based on one $2-D$ and one $3-D$ model and summarised in Table 4.1. The first model features an idealised steel material and a homogeneous geometry. For this case, the homogeneous elasto-plastic behaviour of a typical high-strength steel material is modelled. under any (positive) combination of ϵ_x and ϵ_y, the network shall be able to learn both the interrelation between the responses in the x and y directions (Poisson), as well as the elasto-plastic transition. In order to minimise the amount of inputs and outputs to the ANN, shear is not yet included at this stage.

After tuning the ANN on the simple metal plate, it is time to look at a model with significantly more complexity. A laminated rectangular plate with an elliptical hole is constructed with a Carbon-Fibre Reinforced Polymer (CFRP) composite material. Furthermore, the Hashin criterion is included in the material model in order to introduce the complexity of damage in the mechanical

response, at relatively low computational cost (refer to Subsection 4.2.3. This is elaborated on further in Section 4.2. The modelling space is 3-D due to the nature of the composite Shell elements that require integration points through-the-thickness. However, the loading is confined to the 2-D space with a plane stress condition to limit the amount of ANN inputs and outputs. The loading scenario considered for this model is biaxial tension combined with an in-plane shear stress.

4.1.2. Building the FE Models

In the previous subsection, two benchmark models were chosen. The next step is to actually establish an FE analysis, which is done in Abaqus/CAE [62]. The procedure involves the part creation, element type assignment, mesh generation and the application of boundary conditions. The current subsection elaborates on this process for both models.

(a) (b)

Figure 4.1: An overview of the geometry of the steel plate model (a), where H is the height and W is the width of the plate. A corresponding mesh (b) is constructed exclusively from quadrilateral plane stress elements

High-strength steel plate The first model to be considered is the 2-D rectangular plate made of high-strength steel. It features a simple geometry defined by the width W and height H, as shown in Figure 4.1a. It has no additional features and since the employed material model is purely elasto-plastic, no additional complexity is present in the material either. This model is therefore representing the trivial case of modelling a material response.

Based on that information, it is possible to generate a mesh of any number of elements. In fact, just having one element produces the same results as a more refined mesh such as the one shown in Figure 4.1, because of the simplicity of the model. Only plane strain 4-node quadrilateral element(s) are used, indicated in Abaqus as CPE4.

After defining the mesh, periodic boundary conditions are applied according to the methodology described in Section 4.3. No other boundary conditions are required.

CFRP plate The 3-D composite plate model is different from the steel model in multiple ways. First of all, three extra geometrical quantities are defined. A depth D is assigned through the layup and ply definition definition, which is covered in Subsection 4.2.2. Furthermore, an elliptical hole is present with its vertical and horizontal axis length being defined as E_W and E_H respectively. An overview of the resulting Part object with arbitrary dimensions is provided in Figure 4.2a. The material and section properties are then established and a mesh can be created based on the obtained geometry.

N_{seeds} is defined as the number of seeds on one outer edge of the plate and is assigned to all four edges. The cut-out is seeded with $4 \cdot N_{seeds}$ to obtain a structured mesh with exclusively quadrilateral shell elements, as shown in Figure 4.2b. Since damage is included in this model, high stress-strain gradients are expected locally and therefore triangular elements, which do not posses interpolation capabilities inside the element, would not be a fit choice [62]. Minimum values of N_{seeds} are between 12 and 20, as determined by mesh convergence in Subsection 4.4.3. The element type is chosen to be S4; a 4-node general-purpose shell element [62]. Full integration is

(a) (b)

Figure 4.2: An overview of the geometry of the CFRP plate model (a), where H is the height, W is the width, and D is the depth of the plate. Furthermore, E_W and E_H are the horizontal and vertical dimensions of the elliptical hole. A corresponding mesh (b) is constructed exclusively from 4-node quadrilateral shell elements

used to capture the local stress-strain gradients due to damage as well as possible. The Simpson scheme is used for through-the-thickness integration with three integration points per ply.

After creating the mesh, periodic boundary conditions are applied to the in-plane DoFs according to the methodology described in Section 4.3. Furthermore, the axial out-of-plane DoF of the whole model is constrained with simple supports as it is not within the scope of the current work to study the inclusion of buckling.

4.1.3. Simulation Settings

Both models presented in the previous subsection are set up such that a static analysis step suffices to predict the desired mechanical behaviour. The following settings are chosen.

- The maximum increment size is set to $t_{inc} = 0.001$ with a total step duration of 1. Such a small maximum increment time is required such that enough data points are available on every part of the mechanical response for training the ANN model, even if the FE simulation would converge with a larger increment size.
- For both models, the Full Newton solution technique is employed.
- Non-linear geometry calculations are not required for the high-strength steel model and are thus turned off here. However in the case of the CFRP model, damage effects may cause large local deformations. Consequentially, non-linear geometry calculations are included.
- Adaptive stabilisation is used for the CFRP model analyses, with a maximum ratio of stabilisation to strain energy of $1 \cdot 10^{-4}$. The fraction of dissipated energy shall not exceed 1% of the total energy which is verified in Subsection 4.4.2.

Furthermore, all analyses are performed on the same machine and identical analysis settings as shown in Table 4.2.

Table 4.2: Hardware specification and analysis settings

Machine	CPU type	GPU type	CUDA cores	CPUs per job	GPUs per job
Dell XPS 9560	Intel i7-7700HQ	Nvidia GTX 1050	640	8	1

4.1.4. Post-Processing

Ultimately, the desired output of each simulation is the homogenised response. All behaviour inside the structure is disregarded, which means the post-processing work can be kept compact

and straight-forward.

The first post-processing step is to identify all nodes on the boundary of the structure. Then, the nodal displacements can be extracted for each increment. Axial strains are computed by subtracting the cumulative displacement values of opposing surfaces in each DoF. For an arbitrary increment and DoF i, the the homogenised strain can be expressed as follows.

$$\epsilon_{ii} = \frac{1}{C_i} \left(\frac{\sum_{p=1}^{N_P} u_i^p}{N_P} - \frac{\sum_{q=1}^{N_Q} u_i^q}{N_Q} \right) \tag{4.1}$$

Here, p and q are nodes on opposing surfaces P and Q. respectively, these surfaces contain N_P and N_Q nodes. Furthermore, C_i is the dimension of the model used to normalise displacement to strain as explained in Section 4.3. The equivalent equation for shear strain can be determined by computing the angle γ, as illustrated in Figure 4.3. This angle is computed by creating two vectors $\overline{AB}$ and $\overline{AD}$ and subsequently using the expression for the angle between two vectors.

$$\begin{aligned} \angle BAD = \frac{\pi}{2} - \gamma_{ij} &= \arccos \frac{\overline{AB} \cdot \overline{AD}}{||\overline{AB}||\ ||\overline{BD}||} \\ \rightarrow \epsilon_{ij} = \frac{\gamma_{ij}}{2} &= \frac{\pi}{4} - \frac{1}{2} arccos \frac{\overline{AB} \cdot \overline{AD}}{||\overline{AB}||\ ||\overline{BD}||} \end{aligned} \tag{4.2}$$

The reaction forces are extracted only at the two free-floating nodes. These nodes represent the strain term in the PBC equations, since this is the only point of load application. These forces are then converted to stresses by dividing them in each direction with the corresponding cross-sectional area.

In Subsection 4.1.3, it was stated that the maximum increment size is set to $t_{inc} = 0.001$, effectively resulting in a total of at least 1000 increments per simulation. Since the ANN inputs include the incremental stress and strain it is required to present several data sets with a varying increment size, as motivated in Chapter 3. Therefore, the post-processing phase is performed multiple times for each simulation with varying sample rates in order to obtain the variation in increment size at minimal computational cost.

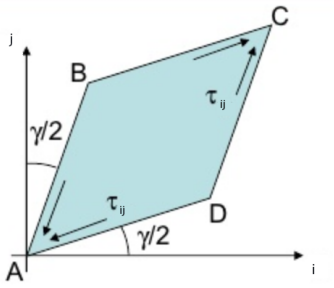


Figure 4.3: Definition of shear strain

4.1.5. Scripted Implementation

Ultimately, the goal of this chapter is to present an automated data generator. As it is rather time consuming to create over a 100 FE models, perform the analyses and post-process by hand, the choice is made to employ the Abaqus/PDE (Python Development Environment) scripting interface. This enables a user to completely parameterise model creation, job execution and post-processing actions. The script is written conforming to Python 2.7.3 conventions, as required

by Abaqus/CAE 2017. An overview of the procedure within the script for the CFRP model is outlined in Algorithm 3, whereas the actual script is presented on Github (`https://github.com/Dwergmaster/ABAQUS_ANN_constitutive_model`). The high-strength steel model is a simplified version of the same code and therefore not elaborated on separately.

Algorithm 3: Algorithm of parametric FE data generator for the CFRP model

Input: W, H, D, E_W, E_H, layup, ply properties, ϵ_{xx}, ϵ_{yy}, ϵ_{xy}, N_{seeds}, t_{inc}
Result: Database with σ_{xx}, σ_{yy}, σ_{xy}, ϵ_{xx}, ϵ_{yy}, ϵ_{xy}, $\Delta\sigma_{xx}$, $\Delta\sigma_{yy}$, $\Delta\sigma_{xy}$, $\Delta\epsilon_{xx}$, $\Delta\epsilon_{yy}$, $\Delta\epsilon_{xy}$ per increment

```
Find all combinations of loads to simulate;
// The following loop iterates over all considered combinations of loading
for load_case in cases do
    Create sketch of W x H rectangle and an ellipse with axes lengths E_W x E_H;
    Define composite layup from layup definition and ply properties;
    Create part, seed outer edges with N_seed and ellipse edge with 4 · N_seeds seeds and
     generate mesh;
    Create step, set maximum increment to t_inc, add stabilisation;
    Create two free-floating nodes for axial and transverse loading and apply PBC
     (according to Section 4.3);
    Apply load_case to the free-floating nodes;
    Create Job object;
    Perform analysis;
    // post-processing starts here
    for sample_rate t_s in rates do
        Initiate data array;
        for each increment in loading step do
            Get current time t;
            if t != n · t_s,   n = 1,2,3... then
                skip to next increment
            else
                Extract reaction forces at free-floating nodes;
                Extract displacements at outer edges;
                Compute homogenised stresses and strains;
                Append current σ_xx, σ_yy, σ_xy, ε_xx, ε_yy, ε_xy, Δσ_xx, Δσ_yy, Δσ_xy, Δε_xx,
                 Δε_yy, Δε_xy to the array;
            end
        end
    end
    Save data array for the current load_case and sample_rate;
end
```

4.2. Material Models

A material model or constitutive model dictates the behaviour of elements in an FEA. Among others, it includes general properties, elastic and/or inelastic mechanical behaviour. The purpose of the material model is to compute the amount of strain (stress-controlled) or stress (strain-controlled) for a given stress-strain state and increment.

The current section covers the material models used for the high-strength steel and the CFRP plates described in this chapter.

4.2.1. High-Strength Steel

The material model for the high-strength steel plate is rather straight-forward, as it is only intended to set up the ANN based model. The material is both homogeneous and isotropic and features the elasto-plastic mechanical behaviour shown in Figure 4.4, based on a typical high-strength steel with properties shown in Table 4.3.

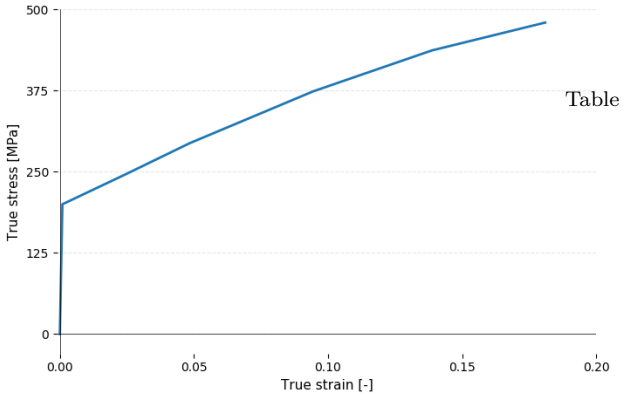


Figure 4.4: Elasto-plastic response of the high-strength steel material model

Table 4.3: Properties of a typical high-strength steel material

Allowable	Value	Sign
E	210	GPa
ν	0.28	-
σ_y	200	MPa
σ_u	480	MPa
ϵ_u	0.181	-

Abaqus requires that all stresses and strains are 'true' values (normalised with instantaneous dimensions), instead of nominal (normalised with initial dimensions). As properties of engineering materials are generally given as nominal quantities, the following conversions are used as presented in Young and Budynas [65]. To get to these equations, it is assumed that plastic deformation is an incompressible event that justifies $l_0 \cdot A_0 = l \cdot A$.

$$\begin{aligned} \epsilon_{true} &= \int_{l_0}^{l} \frac{dl}{l} = \ln\left(\frac{l}{l_0}\right) \\ &= \ln\left(1 + \epsilon_{nom}\right) \\ \sigma_{true} &= \frac{F}{A} = \frac{F}{A_0}\frac{l}{l_0} \\ &= \sigma_{nom}\left(1 + \epsilon_{nom}\right) \end{aligned} \tag{4.3}$$

The difference between true and nominal values becomes apparent after plastic deformation has occurred, as the permanently deformed shape has significantly altered the cross-sectional area and length of the structure at this point.

4.2.2. Carbon-Fibre Reinforced Plastic

The material for a Classical Laminate Theory (CLT) composite FE model is represented by a single orthotropic or fully anisotropic material in Abaqus. This single material is to behave approximately the same as the stack of plies it represents, although certain limits should be acknowledged. A CLT composite model will not be able to capture delaminations and constituent-level failure modes, for instance. Despite its limitations, it is still possible to model ply-level damage and failure phenomena such as matrix cracks and fibre breakage.

To construct the CLT-based material, the anisotropic properties for one ply are first defined as shown in Table 4.4, after which a laminate is built. The local principal axes are indicated by 1 (along the fibres) and 2 (transverse to the fibres). To construct the laminate, one should define the number of plies and the orientation and thickness for each ply. An example of such a laminate is shown in Figure 4.5.

Local stresses and strains in each ply are related to the global stresses and strains through the CLT, as explained by Daniel and Ishai [66]. This theory is a simplified method to construct a

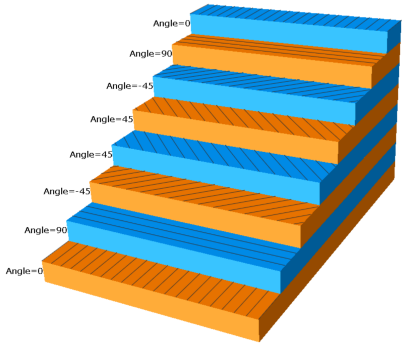


Figure 4.5: Quasi-isotropic layup used for the analyses

Table 4.4: Elastic properties of a typical CFRP ply. Direction 1 is along the fibres and direction 2 is transverse

Elastic		
Allowable	Value	Sign
E_1	161	GPa
E_2	11	GPa
ν_{12}	0.32	-
G_{12}	5.17	GPa
t_{ply}	0.125	mm

direct relation to compute stresses from strains and strains from stresses in a laminate on a global level. It rests on the assumptions that the Kirchhoff-Love theory for bending and stretching of thin plates is valid and the absence of delaminations.

4.2.3. Damage

The term 'damage' is used for a wide variety of phenomena. In a composite laminate, the most frequently encountered damage mechanisms are delaminations, matrix cracks, fibre breakage and fibre/matrix debonding. The scope for this work is to include a damage model that is versatile, efficient, stable and capable of capturing simple in-plane damage. The accuracy compared to actual composites is of lesser importance, as the function of damage is to add complexity to the mechanical response in order to demonstrate the ANN's constitutive modelling capabilities.

With these considerations in mind, several popular FE damage modelling techniques are considered. The work of Chen [67] provides a compact and structured overview of some of these techniques, which is summarised below.

- **Stiffness degradation:** When a chosen damage initiation criterion is satisfied for a certain element, it is assumed that a crack is present. Consequently, the relevant stiffness in this element is reduced such that it is less resistant to loading in certain directions. Then, the load redistribution towards the intact regions is computed. While the stiffness degradation approach is simple, efficient and stable, its applicability to structures with local stress concentrations (such as near a hole or notch) is limited. Near such a stress concentration feature, a coarser mesh causes lower element stresses and therewith, postpones the initiation of damage.
- **Smeared crack:** A useful method in case the modelled crack path is not known in advance is the smeared crack formulation. The basic principle behind this approach is to reduce the stiffness of elements where an energy-based damage criterion is satisfied, which is to represent the crack. Being both robust, efficient and versatile, this technique is suitable to model composites even when complex crack paths are present. A limitation is the mesh dependency of this approach, as the crack (which is a local discontinuity) is always converted to an element-size phenomenon.
- **XFEM:** An alternative approach to FEM with a particular focus on damage modelling is called the eXtended Finite Element Method. It involves the partitioning of an element by introducing additional degrees of freedom and additional 'enrichment' functions to capture the displacement field around a crack. Progressively enriching new nodes prevents the need for adaptive remeshing as the crack path can be described within an element as well. These enrichment functions, which only act on elements that are affected by the crack path or crack tip, have the disadvantage that they add a certain amount of complexity to the shape functions in FEM. Because of this and the non-continuous domain, most applications of

XFEM require dedicated numerical integration techniques which makes the implementation in FEM algorithms challenging.

- **Phantom nodes:** Similar to the XFEM approach, this method considers additional nodes within an element to model a crack. The difference is that with the phantom node method, elements involved directly in the crack field are divided into sub-elements and superimposed. There is no need for modified shape functions, which eases the implementation of this method in FEM software. The numerical integration will have to be performed on partial domains, however, which increases complexity and cost. The nodes of these superimposed elements are called the phantom nodes and allow for modelling the separation.
- **Floating nodes:** This is an approach that uses extra nodes within an element where failure is predicted. Instead of superimposing the complete element, sub-elements are created by partitioning the original element through extra nodes. In contrast to the phantom node method, complex numerical integration over partial domains is not required and it is easier still to integrate in FEM software.

Based on this information and the stated requirements for the damage model, it is decided that the stiffness degradation and smeared crack methods are best in line with the goals of this work. From these two, the smeared crack approach is natively integrated into Abaqus and is therefore the preferred choice.

Damage initiation criterion The approach to model macro-scale laminate damage initiation usually involves the use of the Classical Laminate Theory (CLT) [68]. From a certain global stress-strain state, this theory is employed to compute ply-level stresses and strains. Examples of macroscopic damage initiation criteria (with failure mode indication) of lamina are the maximum strain criterion and Hashin, as explained by Camanho [69], Hashin [70] and Kassapoglou [71]. These criteria both consist of multiple conditions which each represent a failure mode. Firstly, the maximum strain criterion was designed to describe fibre failure, as well as matrix and shear failure. Hashin is based on maximum stresses instead, using modified criteria for fibre failure and the matrix failure by introducing stress interaction terms. It is generally considered to perform better than the maximum strain (or stress) criterion and employed in FE analyses as a simple means of computing damage initiation for UD laminae [62].

$$\left(\frac{\sigma_1}{\sigma^T_{1,u}}\right)^2+\left(\frac{\sigma_{12}}{\sigma_{12,u}}\right)^2\geq 1 \quad \text{if } \sigma_1>0. \text{ Otherwise:} \quad \left(\frac{\sigma_2}{\sigma^C_{2,u}}\right)^2+\left(\frac{\sigma_{12}}{\sigma_{12,u}}\right)^2\geq 1$$

$$\left(\frac{\sigma_2}{\sigma^T_{2,u}}\right)^2+\left(\frac{\sigma_{12}}{\sigma_{12,u}}\right)^2\geq 1 \quad \text{if } \sigma_2>0. \text{ Otherwise:} \quad \left(\frac{\sigma_2}{2\cdot\sigma_{12,u}}\right)^2+\left(\frac{\sigma_{12}}{\sigma_{12,u}}\right)^2+\frac{\sigma_2}{\sigma^C_{2,u}}\left[\left(\frac{\sigma^C_{2,u}}{2\cdot\sigma_{12,u}}\right)^2-1\right]\geq 1 \tag{4.4}$$

In these equations, direction 1 is along the fibres and 2 is the transverse, matrix-dominated direction. In this work, a typical UD CFRP ply is used with its stress allowables given in Table 4.5.

Damage evolution law Now that the damage initiation is defined, a damage evolution law is required to dictate the magnitude of stiffness reduction in each ply. This law is only employed once one of the damage initiation criteria from Equation (4.4) is satisfied and uses damage variables to indicate the degradation of an element's load carrying capability.

Table 4.5: Ultimate strength of a typical UD CFRP ply and damage parameters. Direction 1 is along the fibres and direction 2 is transverse

Allowable	Value	Sign
$\sigma^T_{1,u}$	2800	MPa
$\sigma^C_{1,u}$	1700	MPa
$\sigma^T_{2,u}$	60	MPa
$\sigma^C_{2,u}$	125	MPa
$\sigma_{12,u}$	90	MPa
$G^T_{1,c}$	100	N/mm
$G^C_{1,c}$	100	N/mm
$G^T_{2,c}$	0.22	N/mm
$G^C_{2,c}$	0.72	N/mm

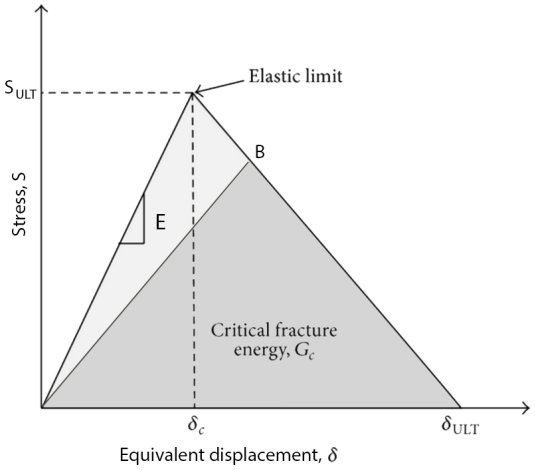


Figure 4.6: Hashin damage evolution model in Abaqus, where the region before δ_c is the simple linear-elastic behaviour and after δ_c, the stiffness reduction due to damage is visualised.

To exert the influence of damage on the finite element model, the fundamental stress-strain relation is reformulated where the elasticity matrix is substituted for the damaged elasticity matrix C_D.

$$\begin{aligned} \bar{\sigma} &= \boldsymbol{C_D}\bar{\epsilon} \\ \boldsymbol{C_D} &= \frac{1}{D}\begin{bmatrix} (1-d_f)E_1 & (1-d_f)(1-d_m)\nu_{21}E_1 & 0 \\ (1-d_f)(1-d_m)\nu_{12}E_2 & (1-d_m)E_2 & 0 \\ 0 & 0 & (1-d_s)G\cdot D \end{bmatrix} \\ \text{where } D &= 1-(1-d_f)(1-d_m)\nu_{12}\nu_{21} \end{aligned} \tag{4.5}$$

The quantities d_f and d_m are the damage variables, with a magnitude between 0 and 1. After damage initiation, they are computed and updated based on an energy criterion which requires the critical energy release rates $G^T_{f,c}$, $G^C_{f,c}$, $G^T_{m,c}$ and $G^C_{m,c}$. All these chosen values are typical for a CFRP composite, while it is noted that $G^C_{1,c}$ is not easily determined experimentally due to fibre kinking. This value is therefore assumed equal to $G^T_{1,c}$, which is acceptable within the scope of this work. The product of the equivalent stress and equivalent displacement is compared to these critical energy levels for each applicable failure mode. A superfluous amount of energy indicates dissipation and moves the elastic limit to a lower stress and higher displacement, illustrated by point B in Figure 4.6.

It was pointed out earlier in this subsection that there is a strong mesh dependency when using the smeared crack approach. Abaqus aims to alleviate this phenomenon by employing a characteristic length in the formulation. This results in a damage evolution law, based on equivalent stress versus equivalent displacement (instead of strain), as shown in Figure 4.6. Essentially, the stresses and strains are converted to equivalent stresses and equivalent displacements, as outlined in Section 21.3.3 of the ABaqus User Manual [62]. The characteristic length is computed based on the element type as well as the element dimensions.

4.3. Periodic Boundary Conditions

The models considered in this work are loaded through the application of axial and shear strain. Since the purpose is to establish a new 'material', the model shall be built as a unit cell which represents the behaviour of an extremely large system, independent of boundary conditions that fully constrain certain degrees of freedom (DoF). It therefore is crucial to apply the boundary conditions in a correct manner to obtain the true material behaviour, by using so called Periodic Boundary Conditions (PBC) [64].

It becomes intuitive to imagine how PBC work by considering a structure with a certain periodicity. When applying a uniform load on this structure, every periodic subsystem or cell experiences the same load and is deformed in the same way as the others. Consequentially, when observing multiple of these elements adjacent to each other, a symmetry exists. The presence of this symmetry dictates that the strains and stresses on a unit cell are identical on all other cells. Thus, a translational condition exists specifically on the boundaries of the cell where it interacts with other cells. Noting that displacements are of the same nature as strains, the following relation can be employed for an arbitrary couple of nodes on two opposing surfaces.

$$A \cdot u_i^P + B \cdot u_i^Q + C_i \cdot \epsilon_{ij} = 0 \tag{4.6}$$

In this equation, P and Q correspond to nodes on two opposing surfaces, i is a chosen DoF and C_i is the geometrical size of the model in the direction of DoF i. Furthermore, A and B are coefficients to be chosen by the user depending on the desired relation between the nodes and ϵ_{ij} is the strain in direction ij (for axial strains, $i = j$). This so-called equation constraint is integrated in Abaqus/CAE and can be called by using the **EQUATION* statement, as described in the Abaqus User Manual [62].

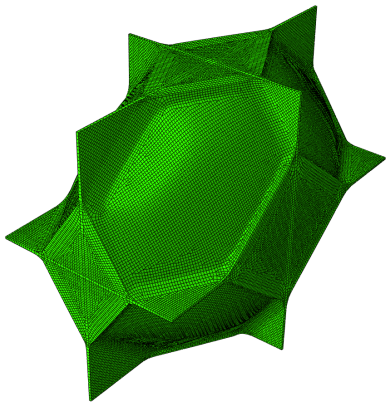


Figure 4.7: FEM model of a generalised 3D unit cell with 6 boundary planes of a closed-cell foam

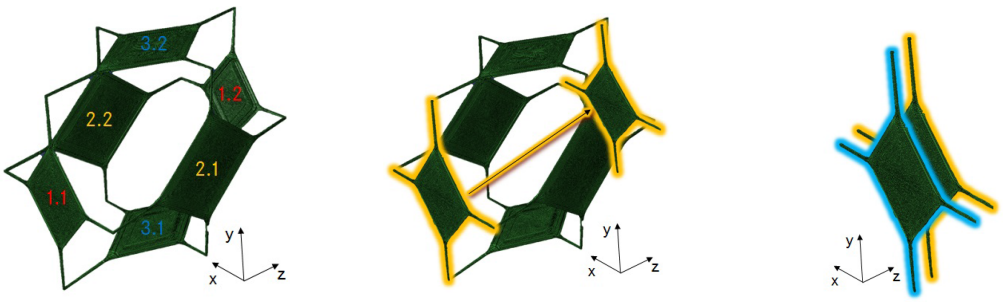
To illustrate the approach, the complexly shaped geometry shown in Figure 4.7 is introduced as it is a non-trivial, 3D example of a PBC application. A similar geometry is employed by Fahlbush [63] to represent the global behaviour of a closed-cell foam by employing just one representative unit cell subjected to PBC. The following paragraphs describe the process of applying the PBC.

The first step in applying the desired boundary conditions is to identify which nodes have to be matched to each other. When providing a complete list of node numbers and nodal coordinates, Algorithm 4 does just that and results in three lists of matching node numbers in each principal direction. A corresponding visualisation of the algorithm is presented in Figure 4.8 for the currently considered example.

The next step is to apply Equation (4.6) and therewith define the actual PBC. The values of A and B are intuitively determined by setting the strain ϵ_{ij} equal to zero. The equation is then

Algorithm 4: Algorithm to find nodal pairs of opposing quadrilateral planes

Data: Node numbers, Nodal coordinates
Result: Lists of nodal pairs $\bar{X}$, $\bar{Y}$, $\bar{Z}$
Find the model's minimum and maximum coordinates in the x, y and z direction;
`// this is where the boundary planes are located`
Filter nodes whose coordinates does not include an extreme value;
Construct 6 sets of nodes, one for each boundary plane (Figure 4.8a);
for *i in [X, Y, Z]* **do**
  Copy nodes in the 'minimum' node list in dimension i and move it to the 'maximum' location of dimension i (Figure 4.8b);
  Find pairs of nodes such that the distance between pairs is minimised (Figure 4.8c);
  Store paired node numbers in an array;
end



(a) Overview of minimum & maximum planes

(b) Copy minimum plane nodes to maximum plane location

(c) Match nodes

Figure 4.8: Node-matching in preparation for periodic boundary conditions

reduced to $A \cdot u_i^P + B \cdot u_i^Q = 0$. As the distance between the opposing nodes should be equal in this case, it follows that $A = 1$ and $B = -1$. A sanity check can be made by looking at the case of axial tension. ϵ_{ii} is positive implying that $u_i^P - u_i^Q$ will have to be equal to $C_i \cdot \epsilon_{ii}$, which is the increase in structure length due to the strain. This enforces that the displacement of each node P will be larger than that of its corresponding node Q in direction i, resulting in an increase in the distance between the nodes.

Another case of interest is shear. Even for this load case, Equation (4.6) succeeds in relating all deformations on the edges of the unit cell. The shear deformation can be applied by taking ϵ_{ij} as shear strain and relating it to the tangential displacements $u_i^P - u_i^Q$. As an example, let us apply a shear in the YZ-plane in the cell of Figure 4.7 and consider the top and bottom plane pair again. A node P on the top will translate towards the positive Y direction while the opposing node Q on the bottom will translate in the equal and opposite direction. This is in line with the equation, which says that $u_i^P - u_i^Q$ should be negative and therefore the location in Y-direction of node Q should be negative compared to node P. This logic can be repeated and proven true for each plane pair and loading condition. Assuming the model has a width W, height H and depth D, the following 3 axial and 6 transverse (shear) equation constraints can be established.

$$
\begin{aligned}
&axial \quad \begin{cases} u_x^{1.2} - u_x^{1.1} + H \cdot \epsilon_{xx} = 0 & \text{for each node pair on planes 1.1 and 1.2} \\ u_y^{2.2} - u_y^{2.1} + W \cdot \epsilon_{yy} = 0 & \text{for each node pair on planes 2.1 and 2.2} \\ u_z^{3.2} - u_z^{3.1} + D \cdot \epsilon_{zz} = 0 & \text{for each node pair on planes 3.1 and 3.2} \end{cases} \\
&transverse \quad \begin{cases} u_y^{1.2} - u_y^{1.1} + W \cdot \epsilon_{xy} = 0 & \text{for each node pair on planes 1.1 and 1.2} \\ u_z^{1.2} - u_z^{1.1} + D \cdot \epsilon_{xz} = 0 & \text{for each node pair on planes 1.1 and 1.2} \\ u_x^{2.2} - u_x^{2.1} + H \cdot \epsilon_{xy} = 0 & \text{for each node pair on planes 2.1 and 2.2} \\ u_z^{2.2} - u_z^{2.1} + D \cdot \epsilon_{yz} = 0 & \text{for each node pair on planes 2.1 and 2.2} \\ u_x^{3.2} - u_x^{3.1} + H \cdot \epsilon_{xz} = 0 & \text{for each node pair on planes 3.1 and 3.2} \\ u_y^{3.2} - u_y^{3.1} + W \cdot \epsilon_{yz} = 0 & \text{for each node pair on planes 3.1 and 3.2} \end{cases}
\end{aligned} \tag{4.7}
$$

An observant reader may have noticed an issue with this set of equation constraints. The corner nodes, which are at the intersection of multiple planes, are over-constrained in the presented formulation. Namely, the axial relations are redundant at those specific nodes and have to be excluded from the set of equations.

The final aspect of the PBC implementation is the application of loads. Currently, the load is represented in the set of Equations 4.7 as the strain ϵ_{ij}. The most straight-forward method of including it in the FE analysis is to create two free-floating nodes, each having 3 DoFs. The DoFs of the first node are then directly referred to as ϵ_{xx}, ϵ_{yy} and ϵ_{zz} in the equations. The second node's DoFs are coupled to ϵ_{xy}, ϵ_{xz} and ϵ_{yz}. This way, a uniform macroscopic load case may be applied to the whole model by simply specifying a displacement on the two nodes for the relevant DoFs.

4.4. Performance and Convergence

The current section evaluates the results of the the FE models described in this chapter. Firstly, the computational cost is analysed in Subsection 4.4.1. Then, the energy levels will be observed to check for numerical issues and other deficiencies of the model in Subsection 4.4.2. Finally a mesh convergence check is performed to determine the required mesh detail in Subsection 4.4.3.

The high-strength steel model is not considered in Subsections 4.4.2 and 4.4.3, as it barely contains any complexity. The outcome of a mesh convergence for this model is simply a set of perfectly overlapping lines and there is no dissipation energy of any type.

4.4.1. Performance and Results

This subsection takes a look at the computational cost of the steel and CFRP model. Furthermore, a simulation of each model will be shown to perform a sanity check on the results.

High-strength steel model The FEA of a steel plate with a plain elasto-plastic material model under biaxial tension is not a costly event. Even though the maximum increment time is limited to 0.001, as motivated in Subsection 4.1.3, one simulation costs only 200 seconds. However, it should be noted that if the maximum increment size is not constrained, one simulation completes in 6 seconds which is considered to be the nominal simulation time.

The outcomes of simulations with this model are straight-forward. Figure 4.9 shows the mechanical response and deformed model for a load ratio of $\epsilon_{xx}/\epsilon_{yy} = 0.5$, showing a constant stress throughout the structure. This specific model has a width x length of 10 mm x 10 mm and is loaded until the ultimate stress of 480 MPa. The mesh parameter N_{seeds} is chosen arbitrarily, but higher than 1 to verify that there is indeed a homogeneous stress field.

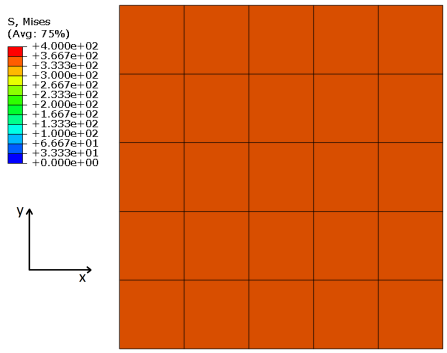


Figure 4.9: Mechanical response of the high-strength steel plate model in biaxial tension with loading ratio $\epsilon_{xx}/\epsilon_{yy} = 0.5$

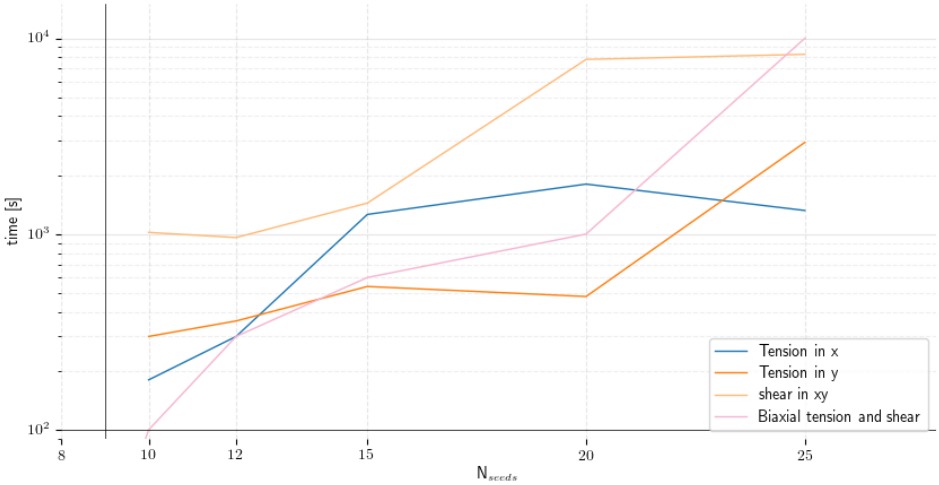
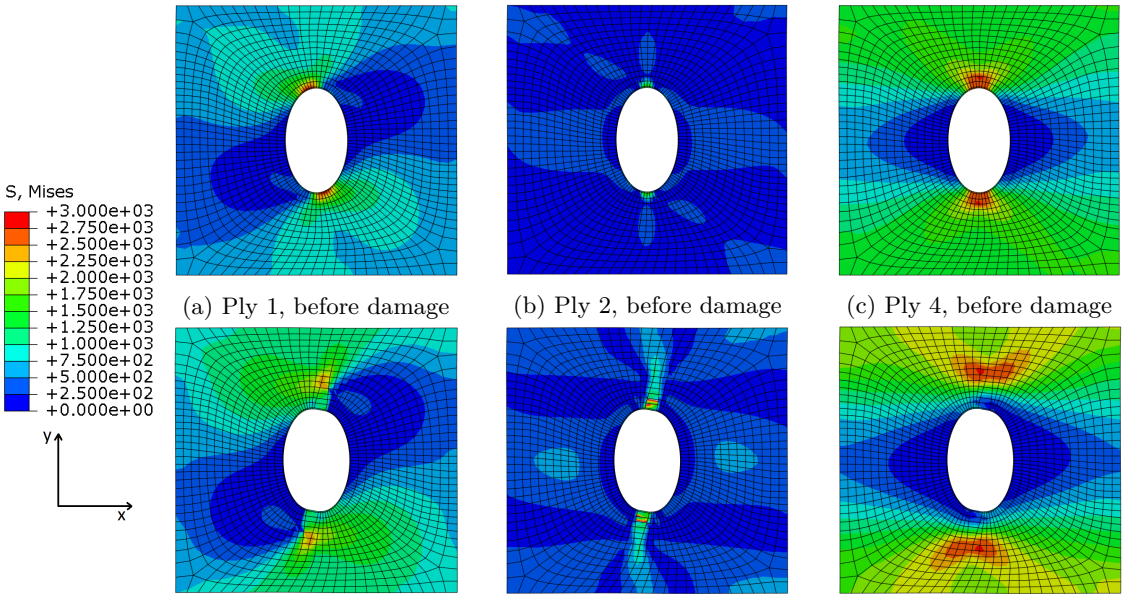


Figure 4.10: Computational cost for several load cases as a function of mesh refinement parameter N_{seeds}

CFRP model The CFRP model is initiated with dimensions $W = 10mm$, $H = 10mm$, $D = 1mm$, $E_W = 2mm$ and $E_H = 4mm$. Furthermore, the lay-up is chosen to be a QI layup with orientations $[45, 90, -45, 0]_s$ with respect to the x-axis that runs along the width dimension. The simulation time is then highly dependent on choice of mesh and loading conditions. By increasing the mesh parameter N_{seeds} for a chosen loading configuration, it is expected that the simulation cost increases. Indeed, it can be seen from Figure 4.10 that this general trend is present. Furthermore, it is apparent that the pure shear load case requires more computational time than uniaxial tension. This can be attributed to the more complex damage state in the model. Uniaxial tension causes a clear load concentration around the hole, giving birth to a clearly defined crack path while the rest of the plate remains undamaged. Shear inflicts damage on the whole plate, requiring significantly more computational time.

A sanity check on the outcomes of simulations of this model is now performed. Figure 4.11 shows the Von Mises stresses in the composite plate loaded in uniaxial strain ϵ_{xx}, for the first, second and fourth ply (45°,90° and 0° w.r.t. the loaded axis). The stress distribution observed here is intuitive; initially (before damage initiation), a clear stress concentration is present around the hole. This is best visualised on ply 4, which carries most loads due to the aligned fibre orientation. Furthermore, ply 2 barely carries loads as the matrix dominated direction is aligned with the load.



(a) Ply 1, before damage (b) Ply 2, before damage (c) Ply 4, before damage

(d) Ply 1, after damage (e) Ply 2, after damage (f) Ply 4, after damage

Figure 4.11: Von Mises stress distribution under a uniaxial load ϵ_{xx}, before and after damage initiation, for ply 1 (45°), ply 2 (90°), ply 4 (0°)

Ply 1 carries a limited amount of loads, but does feature the stress concentration around the hole as well.

After the initiation of damage, the crack front is clearly visualised from the stress distribution in plies 1 and 4. Especially ply 4 shows a typical pattern around the crack front, as compared to the actual crack pattern in Figure 4.12.

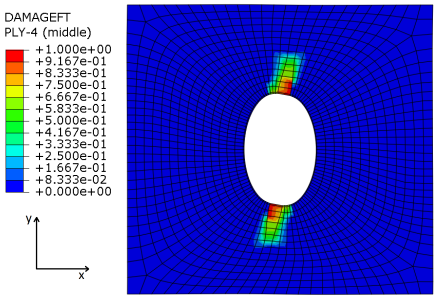
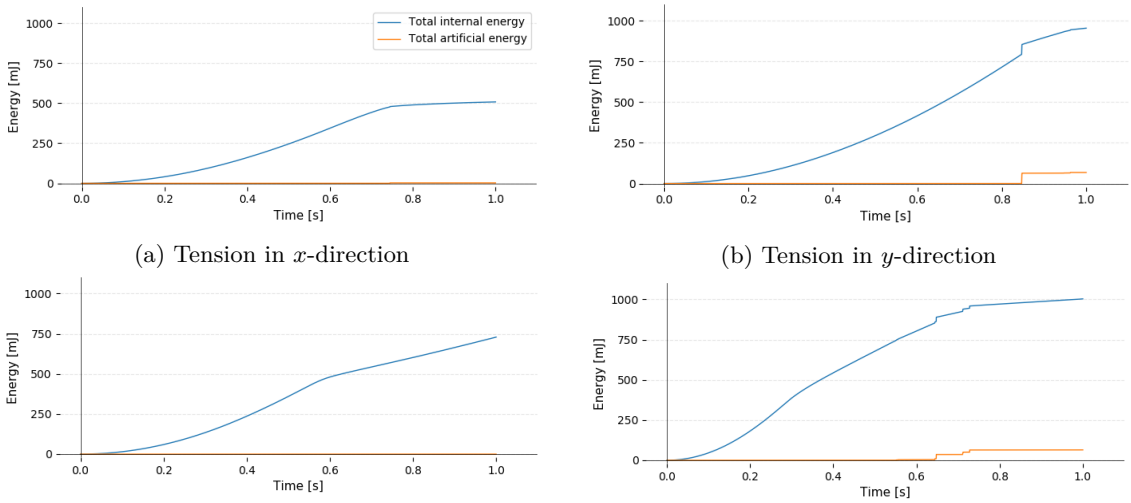


Figure 4.12: Visualisation of the crack front subject to a uniaxial load ϵ_{xx}, after damage initiation

4.4.2. Energy Levels

The presented CFRP model employs artificial damping for both the non-linear geometrical calculations and the damage calculations within the material model. While the addition of damping can aid in reaching convergence during the equilibrium calculations, it may introduce significant amounts of artificial energy and thereby influence the solution. It is therefore good practice to check artificial energy levels, which is done in this subsection. The relevant energy levels are the following.

- Total internal energy: the total energy present in the model, accumulated from elastic strain energy and the energy dissipated by plasticity;



(a) Tension in x-direction

(b) Tension in y-direction

(c) Shear on xy-plane

(d) Combined biaxial tension and shear

Figure 4.13: Energy levels in the CFRP model for several loading scenarios with $N_{seeds} = 15$

- Total artificial energy: the energy used to stabilise equilibrium calculations. This is particularly useful and sometimes required when damage and/or other non-linear phenomena are included in a model. A rule of thumb is that this value should not rise above 5% of the total internal energy;

Figure 4.13 provides an overview of these energy levels for a mesh size of $N_{seeds} = 15$. It is observed that for all cases, the total artificial energy is well below 5% of the internal energy. Furthermore, the static dissipation below 0.5% of the total internal energy for all cases and barely visible in the plots.

4.4.3. Mesh Convergence

Almost any FE model is subject to mesh dependencies. In other words; the choice of mesh significantly influences the outcome of a FEM simulation. Intuitively, one would expect that a finer mesh results in a more accurate solution. Computational time increases simultaneously however, so a compromise has to be made between simulation accuracy and cost. To determine the desired refinement, a mesh convergence is carried out in this subsection.

Toperform the mesh convergence, mechanical responses of four select loading scenarios are visualised in one figure for multiple values of mesh parameter N_{seeds}, which specifies the amount of nodes on each outer edge. The results for uniaxial tension in the x and y direction, as well as shear loading in the xy plane are visualised in Figures 4.14 to 4.16. Furthermore, the case of combined loading with equal loading ratios is visualised in Figure 4.17. Clearly, the mesh has a visible influence on the mechanical response. The following observations are made with respect to the results of $N_{seeds} \in [10, 25]$.

- Negligible difference in initial structural stiffness;
- Peak strength tends to decrease with an increasing mesh refinement due to earlier initiation of damage. This is intuitive, as smaller elements at the edge of the cut-out experience a higher average stress and tend to fail earlier;
- Response after damage initiation is heavily mesh-dependent. In general, it is observed that a coarse mesh results in a sudden failure of the global structure whereas a fine mesh facilitates a more gradual failure;
- The mesh dependency is visible in particular for pure shear loading, where the strength in the damaged regime significantly increases with a finer mesh. This is attributed to the

absence of delaminations that introduce a size effect when using CLT, as explained in Chen et al. [72].

Based on the findings concerning the mesh convergence, the simulation cost and the energy levels, it is decided to use a mesh parameter of $N_{seeds} = 15$ for the CFRP model simulations in this work.

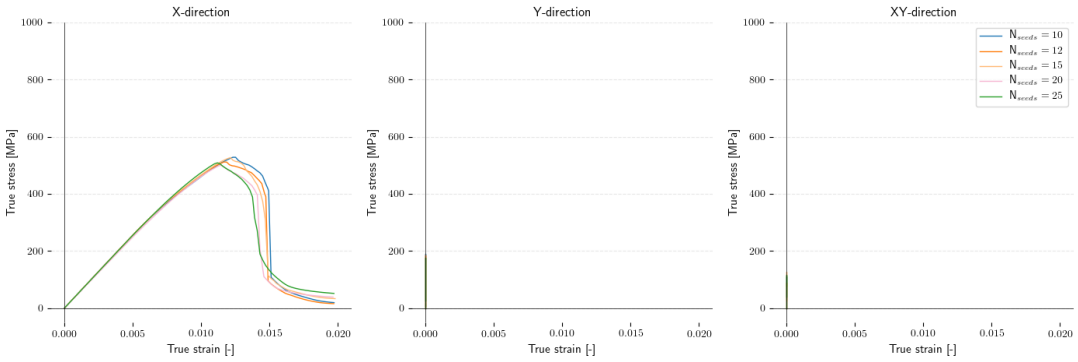


Figure 4.14: Mesh convergence for uniaxial tension (x-direction)

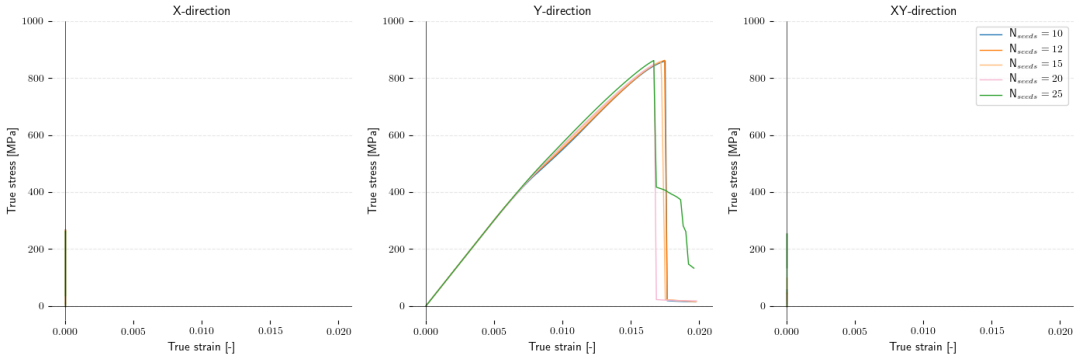


Figure 4.15: Mesh convergence for uniaxial tension (y-direction)

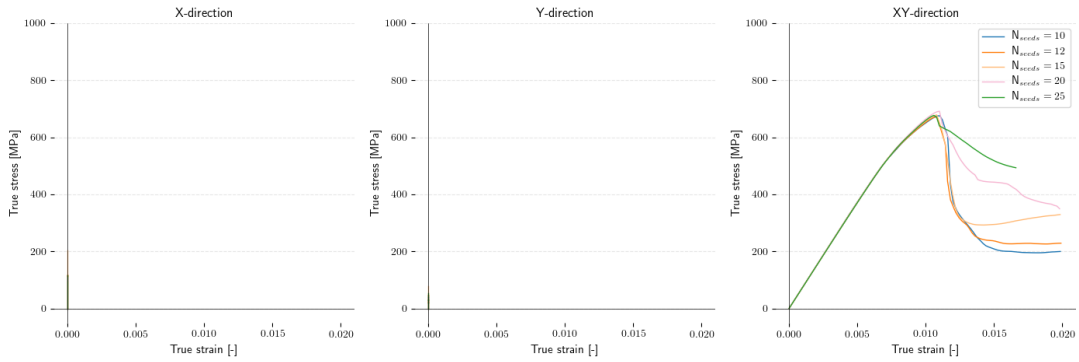


Figure 4.16: Mesh convergence for shear (xy-plane)

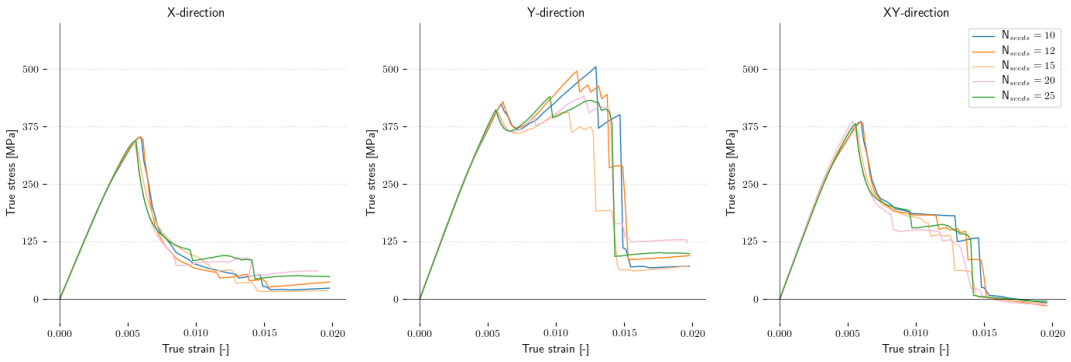


Figure 4.17: Mesh convergence for biaxial tension (x and y-direction) and shear (xy-plane)

4.5. Summary

The current chapter aimed to detail the design of the Finite Element (FE) data generator and its scripted implementation. Four major aspects were considered: the choice of the benchmark cases for this thesis, the employed material models, the boundary conditions and the performance and convergence of each model.

Due to the novelty of the currently proposed framework, there is no literary source available on which the initial Artificial Neural Network (ANN) design could be based. Therefore, it was decided to set up two FE models to facilitate a quick and efficient design process: one computationally cheap and simple case followed by a more expensive and complex case. The first case was chosen to be a homogeneous elasto-plastic high-strength steel material without geometrical features. In contrast, the second case was chosen to be a plate with an elliptical cut-out consisting of a quasi-isotropic Carbon Fibre Reinforced Composite (CFRP) material. The CFRP material was modelled based on the classical laminate theory, therewith omitting delamination and constituent-level phenomena. Based on a comparison between damage modelling methods in FEM, the smeared crack approach (based on Hashin damage) was chosen. This increased the complexity and capability of the model by including damage propagation and failure by fibre breakage and matrix cracks.

As to provide a suitable set of boundary conditions for simultaneous biaxial and shear loading as well as facilitate the extraction of homogenised stresses and strains, it is chosen to implement periodic boundary conditions. By doing this, the degrees of freedom of corresponding nodes on opposing boundaries were coupled to a free-floating node. Two of these free-floating nodes exist, representing the axial and shear directions respectively. This facilitates an intuitive way of applying a set of loads, as well as a straight-forward post-processing procedure to obtain the homogenised stresses and strains.

Finally, the performance of the CFRP model was verified for several mesh refinement settings. Based on the findings concerning the mesh convergence, the simulation cost and the energy levels, the mesh was chosen.

5

Analysis and Results

At this point, all aspects of the proposed constitutive modelling framework based on an Artificial Neural Network (ANN) have been covered. The current chapter applies the established procedure to the two 2-D FEM models described in Section 4.1. The results are presented with the help of the mechanical responses of the ANN constitutive model applied to a verification data set.

The considered models are treated in a sequence of increasing complexity, starting with a simple biaxially loaded steel plate with a purely elasto-plastic response in Section 5.1. The next step is a Quasi-Isotropic (QI) composite plate with an elliptic hole and a material model including in-plane damage phenomena, covered in Section 5.2. This plate is loaded under biaxial tension and shear and will serve as the final demonstration of the methodology described in this thesis. Finally, a summary is provided in Section 5.3.

5.1. Modelling an Elasto-Plastic Steel Plate

While being relatively simple, the goal of applying the methodology to the elasto-plastic high-strength steel plate is to establish the model and obtain a good initial guess at the many hyperparameters. This implicit, static and 2-D Finite Element (FE) model allows for quick data generation and due to the simple shape of the mechanical response curves, it facilitates quick ANN training as well.

5.1.1. Design of Experiments

The generation of data for the ANN is the first challenge to overcome. As a sufficiently representative data set is required, it is first identified what simulations are to be performed. With the FE model layout and elasto-plastic material properties already outlined in Sections 4.1 and 4.2, it is only required to define the set of dimensions and external conditions. The dimensions are chosen as a simple square, defined by $[W, H] = [10, 10]\ mm$. To set up the external conditions, let us first identify the independent ANN input variables.

- Load in x-direction;
- Load in y-direction;
- Load increment magnitude.

With this in mind, a grid of simulation settings needs to be generated. The data generation is, in fact, the major bottleneck of the proposed framework and thus it is important to reduce the amount of required simulations to a minimum. As pointed out in Subsection 3.1.1, one essential advantage is that the increment magnitude can be omitted as independent variable. As explained

in Subsection 4.1.4, one can simply extract data with multiple increment magnitudes from one simulation that is performed with a sufficiently low maximum increment time. Thus, only two independent variables are present for the 2-D biaxial case considered here.

To construct the grid of settings, multiple grid spacings are investigated. From this it is determined that the best way to construct the grid is to define the minimum and maximum value for each variable and have at least 4 linearly spaced steps in total. The minimum and maximum load values are chosen as 0 and $\epsilon_{x,y} = 0.18$ respectively, with a stopping criterion in place for when the maximum combined load is reached. No compression is considered as this would require the consideration of buckling, which cannot be done since the current model is 2-D. Furthermore, the additional complexity that buckling brings is not within the scope of this thesis. Summarising, the following grid is formed.

$$\epsilon_x = 10^{-2} \cdot [0.0, 6.0, 12, 18]$$

$$\epsilon_y = 10^{-2} \cdot [0.0, 6.0, 12, 18]$$

$$\Delta\epsilon = 10^{-4} \cdot [4.0, 8.0, 16, 32]$$

Setting up all combinations results in 15 (The all-zero load case is not considered) FE simulations and 60 data files which serve as the training data set for the ANN.

5.1.2. Hyperparameters and Results

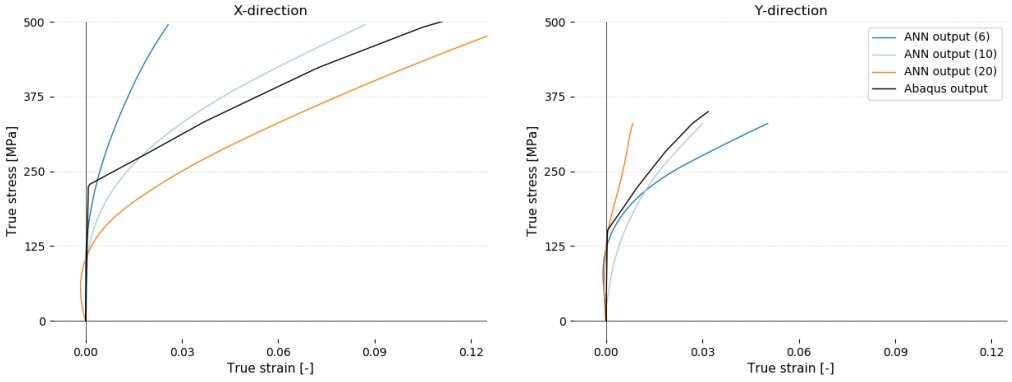
The process of fine-tuning the hyperparameters has been outlined in Section 3.2. Below, the actual process of finding these parameters is performed by using one verification data set with loading ratio ($\epsilon_x/\epsilon_y = 18/12$). To perform the verification of the final choice of hyperparameters, an additional verification case is considered with loading ratio ($\epsilon_x/\epsilon_y = 6/18$). Concerning the ANN set-up, it is decided that only the *baseline* design is used for the current case.

Architecture The amount of Hidden Layers (HL) and nodes per layer are optimised first, while keeping the training settings at its default value for now. First, only one HL is used to model the response. The amount of nodes are gradually increased and it is found that when $N_1 \leq 5$, the ANN is not able to capture the response properly. When further increasing the amount of nodes, it is observed that the elastic-plastic transition in the curve is captured better (by a 'sharper' transition). Despite this, even a 1-HL 20-node ANN can only capture the full response with moderate accuracy as shown in Figure 5.1a. In fact, after increasing the amount of nodes above 10, the response is worse for the same training time.

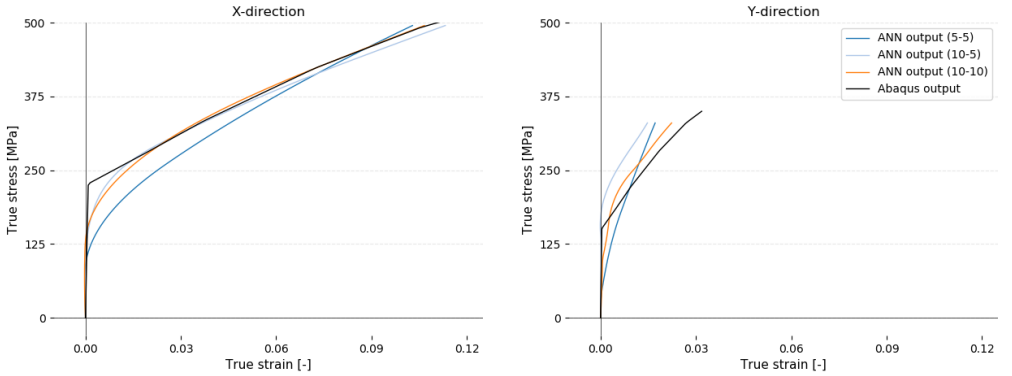
Now, the amount of HLs is increased to two. Even an ANN structure of $N_{1,2} = (5, 5)$ is able to capture the response better than the 1-HL network, as seen from Figure 5.1b. The $N_{1,2} = (10, 10)$ network fits the desired mechanical response quite well already. When further increasing the amount of HLs and nodes per layer, no improvement is observed for the current model.

Training time To further improve the ANN model, the training time can be increased. The amount of optimisation loops is a measure for training time and indicated with N_{epochs}. Figure 5.2 shows a comparison of several ANN responses as a function of epochs, from which improvement is indeed observed when a longer training time is used. When increasing the amount of epochs above 1500, however, there is no additional improvement and the ANN has converged.

Initialiser The choice of initialiser is determined by training the ANN for 500 epochs on standard optimisation settings, for a 10-10 layer architecture. No clear conclusion can be drawn from Table 5.1 as the performance of the ANN seems to be similar for most initialisation schemes, except the *random*-based ones. For now, *glorot_normal* is the scheme of choice and the performance will be re-evaluated in Section 5.2.



(a) 1 HL



(b) 2 HL

Figure 5.1: ANN response with 1 and 2 HLs for the steel plate model. The legend indicates the amount of nodes in each HL

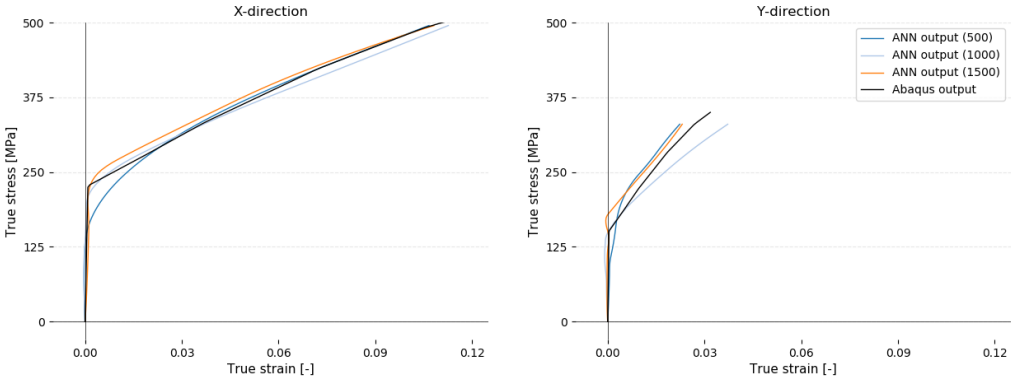


Figure 5.2: ANN response for several training times for the steel plate model. The legend indicates N_{epochs}

Optimisation parameters The optimisation algorithm is chosen by evaluating several of the most well-known schemes with their standard settings. A 2-HL ANN with a $N_{1,2} = (10, 10)$ architecture is initialised with a *glorot_normal* scheme and trained for 500 epochs, after which the accuracy is evaluated. From the overview in Table 5.2, it can be concluded that the *Adadelta*, *Adam* and *Nadam* are all good choices. However, in practice it is found that the *Nadam* scheme is the most dependable choice with the least amount of randomness in the results.

For this scheme, it is iteratively determined what the best choices are for the learning rate α,

Table 5.1: training Performance comparison of popular initialisers (covered in Subsection 2.2.5) after training for 500 epochs. All other hyperparameters are kept constant

Initialiser	Accuracy [%]
random_uniform	71.8
random_normal	73.9
lecun_uniform	76.4
lecun_normal	75.3
he_uniform	76.3
he_normal	77.6
glorot_uniform	75.4
glorot_normal	77.8

Table 5.2: Performance comparison of popular training schemes after training for 500 epochs with recommended optimiser settings. All other hyperparameters are kept constant

Optimiser	Accuracy [%]
Stochastic Gradient Descent (SGD)	75.7
Root Mean-Squared Propagation (RMSprop)	80.8
Adaptive Gradient Descent (Adagrad)	86.0
Adagrad with restricted gradients (Adadelta)	91.3
Adaptive Momentum Estimation (Adam)	91.2
Nesterov-accelerated Adaptive Moment Estimation (Nadam)	90.6

decay γ, as well as the β_1 and β_2 terms. The following (range of) values are found to work well, resulting in an ANN response of two verification cases shown in Figure 5.3.

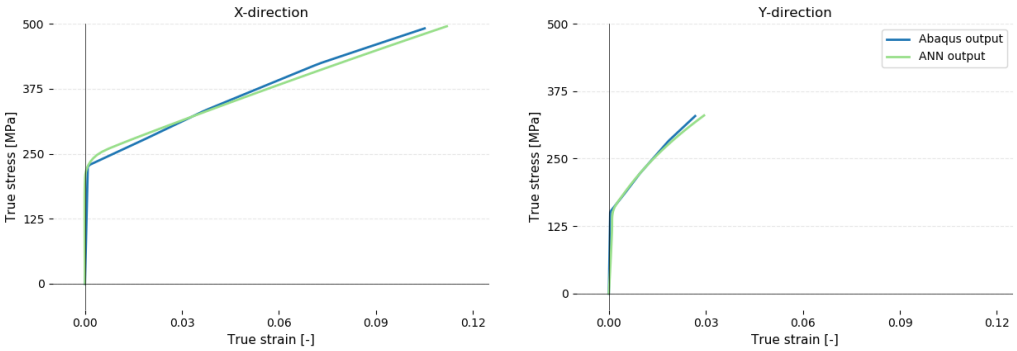
- α: 0.005 - 0.01
- γ: 0.001 - 0.01
- β_1: 0.9 - 0.95
- β_2: 0.999

5.1.3. Performance and Discussion

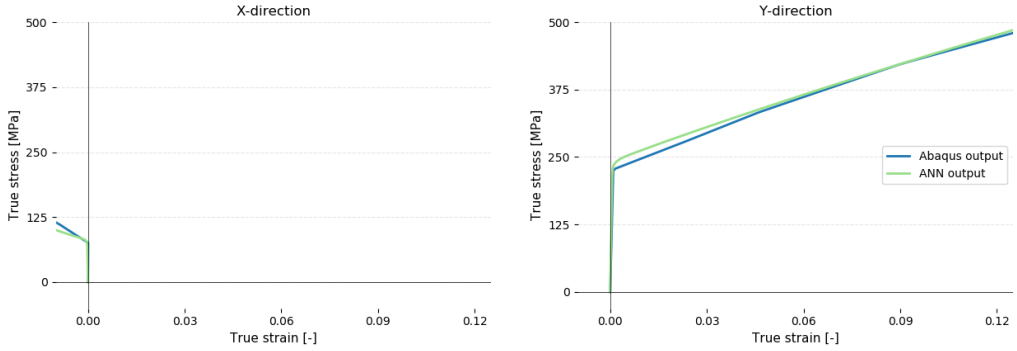
The high-strength steel plate model considered in this section represented the trivial case of capturing an elasto-plastic constitutive model, without the influence of geometrical features. Despite being a simple model, this step proved useful to set up an initial ANN model and find a suitable combination of the many involved hyperparameters.

Firstly, the data generating step was performed on a 4x4 grid of tensile loads in the x and y direction. This resulted in 15 simulations which were each evaluated for 4 load increment sizes, resulting in 60 data files. It was found that a load grid with less than 4 steps is not sufficient to capture the constitutive behaviour properly.

Subsequently, the ANN was initiated and optimised by determining a suitable set of hyperparameters. This was done in steps by sequentially determining an appropriate architecture, training time, initialiser and optimisation settings. Finally, an ANN model was obtained that was demonstrated to be able to make excellent predictions of two chosen verification cases, which were not part of the training data. The observed maximum error in stress was below 5% in all cases. With this, it is deemed capable of successfully representing the constitutive model and mechanical response of the steel plate.



(a) Loading ratio $\epsilon_x/\epsilon_y = 18/12$



(b) Loading ratio $\epsilon_x/\epsilon_y = 6/18$

Figure 5.3: ANN response for the steel plate model of two verification cases with the final choice of hyperparameters

An overview of the final performance of the framework proposed in this thesis, applied to the steel plate model, is provided in Table 5.3. The ANN-based FE model is identical to the conventional steel plate in layout, but its material model is substituted with the trained ANN.

Table 5.3: Computational time of each module in the framework for the steel plate model

Action	time [s]
Data set generation (15 analyses)	300
ANN training	15
ANN-based FE model	3

5.2. Modelling a Composite Plate with an Elliptic Cut-Out

Having established an initial ANN constitutive model and ensured the functionality of each module within the proposed framework, this section aims to explore the model's capabilities to capture more complex shapes. The ultimate goal is to establish a mesh coarsening approach while maintaining the accuracy of the fine mesh. The composite plate with an elliptic cut-out introduced in Section 4.1 does just that, as it is characterised by both complex material behaviour and a structural feature that influences the mechanical response.

5.2.1. Design of Experiments

The established FE model is analysed by means of an implicit, static analysis. It is based on the design shown in Subsection 4.1.2, where the dimensions are chosen as $[W, H, D, E_W, E_H] = [10, 10, 1, 2, 4]$ mm. Despite its 3-D set-up, it is only loaded in-plane in biaxial tension and shear. The reason for not considering compression loading is the risk of buckling, which is beyond the scope of this thesis. It is pointed out that even a shear load may result in buckling, which is a limitation of the current version of the model implying that the results shouldn't be considered physically accurate when shear is the major loading mode. The independent ANN input variables for the current problem are the following.

- Load in x-direction;
- Load in y-direction;
- Load in the xy-plane;
- Load increment magnitude.

As pointed out already in Subsection 5.1.1, the increment magnitude can be omitted when performing simulations with a sufficiently small maximum increment size. Thus, the grid of simulation settings is a combination of the three loadings. For the steel plate model, it was found that 4 linearly spaced steps are sufficient to construct the grid and this is found suitable for the current case as well.

$$\epsilon_x = 10^{-3} \cdot [0.0, 6.67, 13.3, 20.0]$$

$$\epsilon_y = 10^{-3} \cdot [0.0, 6.67, 13.3, 20.0]$$

$$\epsilon_{xy} = 10^{-3} \cdot [0.0, 6.67, 13.3, 20.0]$$

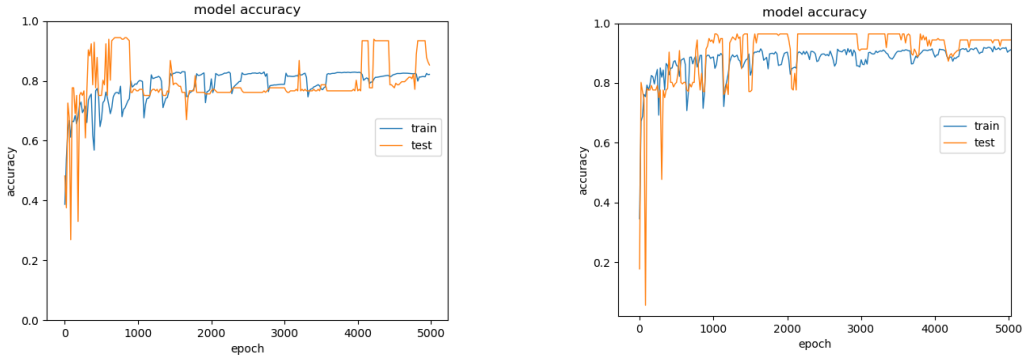
$$\Delta\epsilon = 10^{-5} \cdot [1.0, 5.0, 15, 20]$$

As the current simulations already take significant simulation time, it is decided to try to achieve a further reduction in required simulations by eliminating any entry that does not contain a maximum load. This way, it is ensured that the full damaged region is modelled for every simulation and data file. By doing this, the amount of grid points (simulations) are reduced from 63 to 37. Again, post-processing is performed on every grid point for 4 load increment magnitudes thus effectively resulting in 148 data files that are part of the training data set.

5.2.2. Hyperparameters and Results

Similar to the steel plate, the hyperparameters of the ANN model have to be tuned such that a high-accuracy response is obtained. The optimal settings found in Subsection 5.1.2 will serve as 'initial guess', from which iterations are performed on several parameters. During this process, the ANN performance is measured by evaluating a verification case with loading ($\epsilon_x/\epsilon_y/\epsilon_{xy} = 13.3/20.0/6.67$). The final configuration of hyperparameters following from this subsection is verified by considering two additional (randomly chosen) cases, namely ($\epsilon_x/\epsilon_y/\epsilon_{xy} = 0.0/0.0/20.0$) and ($\epsilon_x/\epsilon_y/\epsilon_{xy} = 6.67/13.3/20.0$).

For now, only the baseline ANN is considered whereas the other three proposed configurations are covered in Subsection 5.2.3. The ANN architecture and training time will be evaluated using the visual response of the verification case, as these two hyperparameters are the main factors that influence the complexity and non-linear response the network can capture. After finding proper settings for these, the choices regarding the initialiser and optimiser can be determined by a quantitative comparison of the verification accuracy.



(a) $N_{1,2} = (10, 10)$ (b) $N_{1,2,3} = (40, 40, 40)$

Figure 5.4: Accuracy during the training process of a 2-HL and 3-HL ANN

Architecture To determine the architecture, all initial settings are set to the final ones found for the steel model except for the training time. N_{epochs} is set to 5,000 for now, as this number results in a reasonable convergence for both small and large networks as seen from Figure 5.4. While this training time is enough for a qualitative comparison between architectures, an ideal choice for this hyperparameter will be determined still.

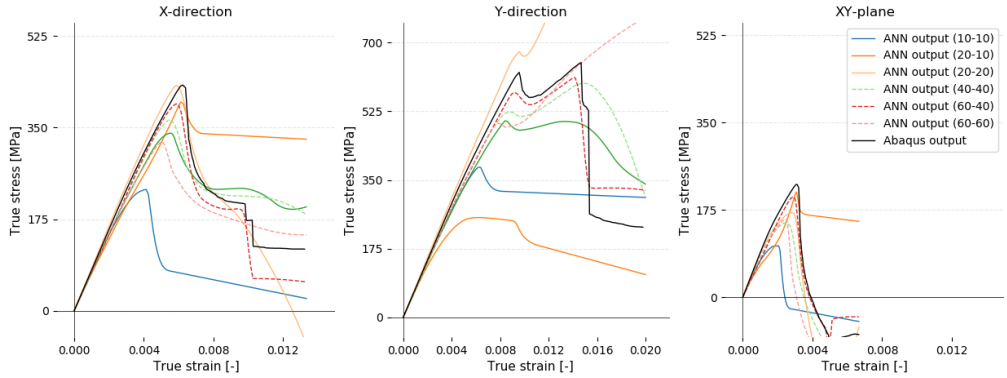
The considered ANN architectures feature 2 and 3 HLs respectively. A visual evaluation of the chosen verification case is employed, as it allows for a clear indication of how well the ANN is able to capture the complexity of the composite plate to a sufficient degree. A comparison between various node configurations is shown in Figure 5.5. Here, it is seen that a 3-HL ANN does not provide any improvement over the 2-HL configuration and the best fit is observed with a 2-HL network with an $N_{1,2} = (60, 40)$ configuration.

Training time Choosing a proper training time, measured in the amount of epochs, can increase the model accuracy. Figure 5.6 provides an overview of the verification response for multiple values of N_{epochs} for an $N_{1,2} = (60, 40)$ architecture. Similar to the comparison of architectures, a visual comparison is more useful here than just using numerical accuracies to determine how well the ANN captures non-linearities. A clear convergence is observed when increasing N_{epochs} towards 10,000. After this amount of training, no significant improvement is seen when further increasing the training time to 20,000. Thus, $N_{epochs} = 10,000$ is considered to be the converged point in the training process and is the final choice.

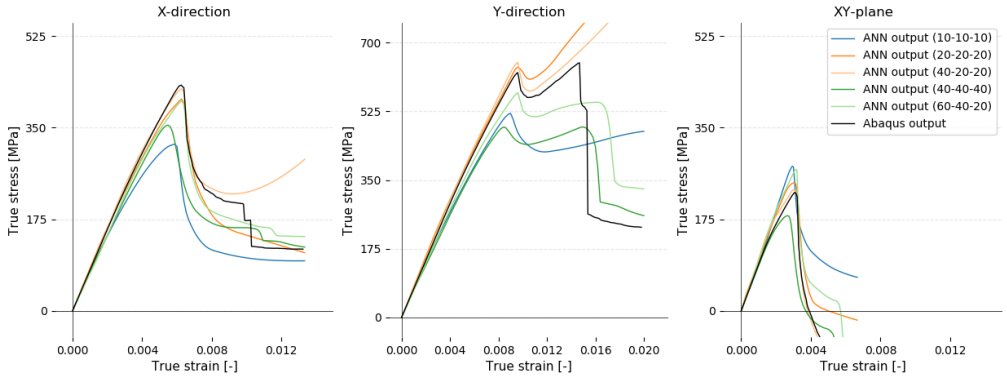
Table 5.4: Performance comparison of popular initialisers (covered in Subsection 2.2.5) after training for 10,000 epochs. All other hyperparameters are kept equal

Initialiser	Accuracy [%]
random_uniform	89.2
random_normal	91.7
lecun_uniform	94.0
lecun_normal	94.0
he_uniform	91.32
he_normal	91.67
glorot_uniform	93.0
glorot_normal	93.2

Initialiser For the current case, the popular initialisers considered in Subsection 5.1.2 are evaluated again. The resulting training accuracies are shown in Table 5.4, which are in line with the



(a) 2 HL



(b) 3 HL

Figure 5.5: ANN response with 2 and 3 HLs for the composite plate model. The legend indicates the amount of nodes in each HL

findings for the steel model in the sense that they all perform quite well. No visual comparison is required as the initialiser settings do not influence the complexity that can be modelled by the ANN. From the found accuracies, it is observed that the best choices for the composite plate model are *lecun_uniform* and *lecun_normal*.

Table 5.5: Performance comparison of popular training schemes after training for 10,000 epochs with recommended settings. All other hyperparameters are kept equal

Optimiser	Accuracy [%]
Stochastic Gradient Descent (SGD)	76.5
Root Mean-Squared Propagation (RMSprop)	90.2
Adaptive Gradient Descent (Adagrad)	89.2
Adagrad with restricted gradients (Adadelta)	82.8
Adaptive Momentum Estimation (Adam)	91.5
Nesterov-accelerated Adaptive Moment Estimation (Nadam)	93.4

Optimisation parameters For the steel plate model, three optimisers were identified to be a good choice. The results are in line with the findings for the composite plate presented in Table 5.5, based on which the choice for the *Nadam* optimiser is reinforced.

Apart from the training scheme, it is essential to re-investigate the optimisation parameters α, γ, β_1 and β_2. The initial choice is the set of values determined in Subsection 5.1.2, which are

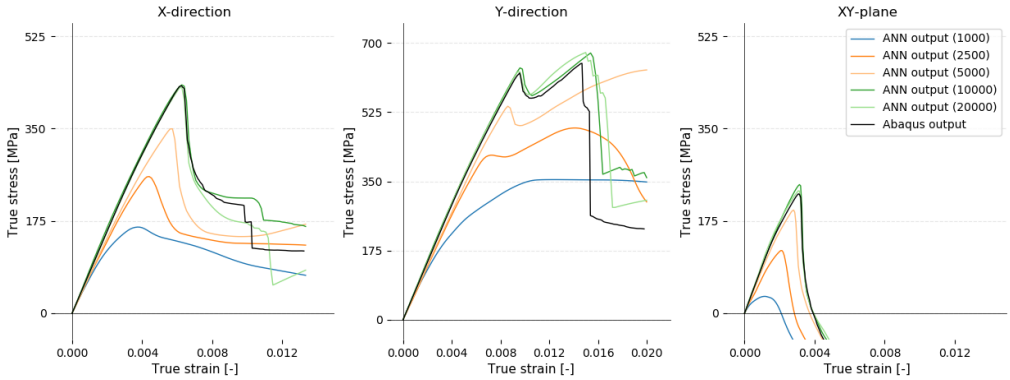


Figure 5.6: ANN response for several training times for the composite plate model. The legend indicates N_{epochs}

varied one by one to investigate their influence on the response. It is found that the following (range of) values result in the best fit, which is demonstrated by the three verification cases shown in Figure 5.7.

- α: 0.008
- γ: 0.008
- β_1: 0.95
- β_2: 0.999

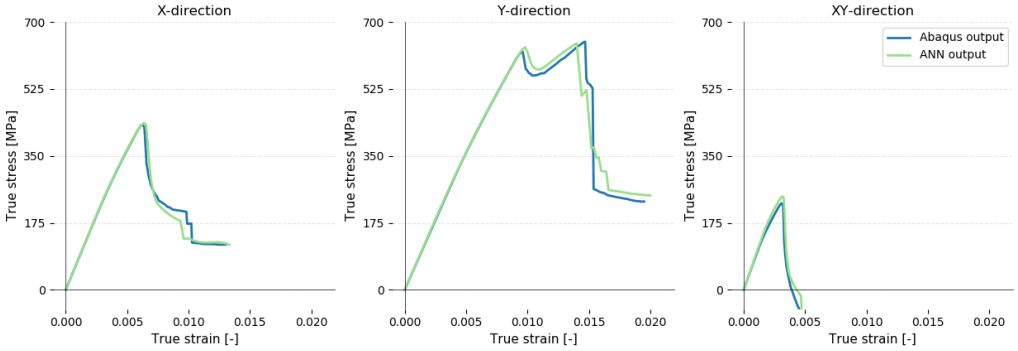
5.2.3. Comparison of Input/Output Configurations

In Section 3.2, four configurations of ANN inputs and outputs are proposed. While the current section has only considered the *Baseline* configuration so far, this subsection investigates the performance three alternative options. For each one, the hyperparameters are first tuned according to the same methods found in Subsection 5.2.2 after which the final ANN model is obtained.

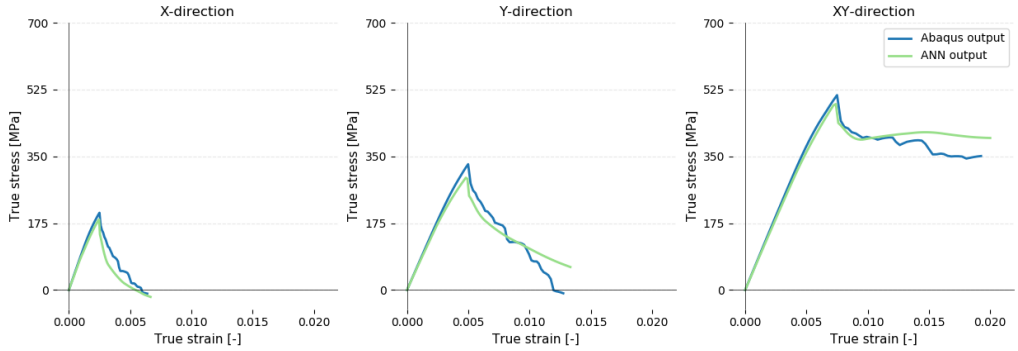
Two means are used to compare these performances. First of all, a plot of the training and testing accuracy is provided in Figure 5.8 to illustrate the convergence during the training process. Secondly, the three chosen verification cases are evaluated and compared for each of the configurations in Figure 5.9.

Including health parameters Based on the current stress-strain state, a set of health parameters can be computed and fed into the ANN as explained in Subsection 3.2.2. While this is a derived input, it provides the ANN with a measure of the initial stiffness value in relation to the instantaneous secant stiffness. This addition to the ANN inputs does not promote convergence or accuracy, however. As seen from Figure 5.9, the contrary is true and the ANN seems to have a clear tendency to delay and reduce the effect of damage. This is an interesting finding as the idea behind this proposed configuration was to provide a better estimation of the damaged response region. Despite the use of a relatively expensive hyperparameter configuration, the ANN is not able to establish a good fit.

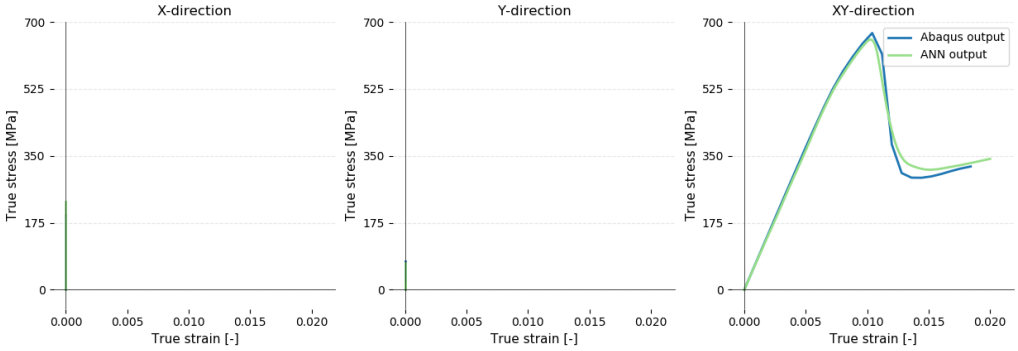
- $\alpha = 0.008$, $\gamma = 0.008$, $\beta_1 = 0.95$, $\beta_2 = 0.999$
- $N_{epochs} = 20,000$
- $N_{1,2} = (60, 40)$
- Initialiser = *Lecun uniform*



(a) Loading ratio $\epsilon_x/\epsilon_y/\epsilon_{xy} = 13.3/20.0/6.67$



(b) Loading ratio $\epsilon_x/\epsilon_y/\epsilon_{xy} = 6.67/13.3/20.0$

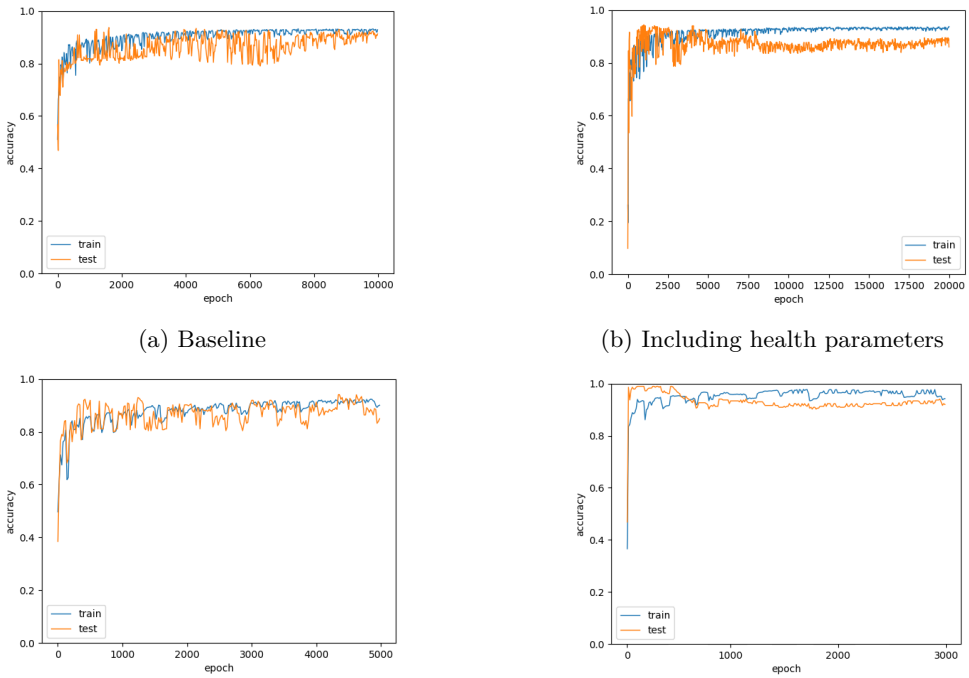


(c) Loading ratio $\epsilon_x/\epsilon_y/\epsilon_{xy} = 0.0/0.0/20.0$

Figure 5.7: ANN response of three verification cases for the composite plate model, with the final choice of hyperparameters

Including historic stress-strain states The inclusion of previous stress-strain states provides the ANN with the means to use the gradient of the mechanical response to predict the next stress increment. From the verification cases shown in Figure 5.9, it is observed that this modification does not significantly improve the response, although the ANN converges within less epochs as compared to the baseline configuration. The following hyperparameters are ultimately employed for the ANN with a historic stress-strain state input.

- $\alpha = 0.008$, $\gamma = 0.008$, $\beta_1 = 0.95$, $\beta_2 = 0.999$
- $N_{epochs} = 5,000$
- $N_{1,2} = (60, 40)$
- Initialiser = *Lecun uniform*



(a) Baseline

(b) Including health parameters

(c) Including historic stress-strain states

(d) Minimum input

Figure 5.8: Training and testing accuracy during training for several ANN input/output configurations

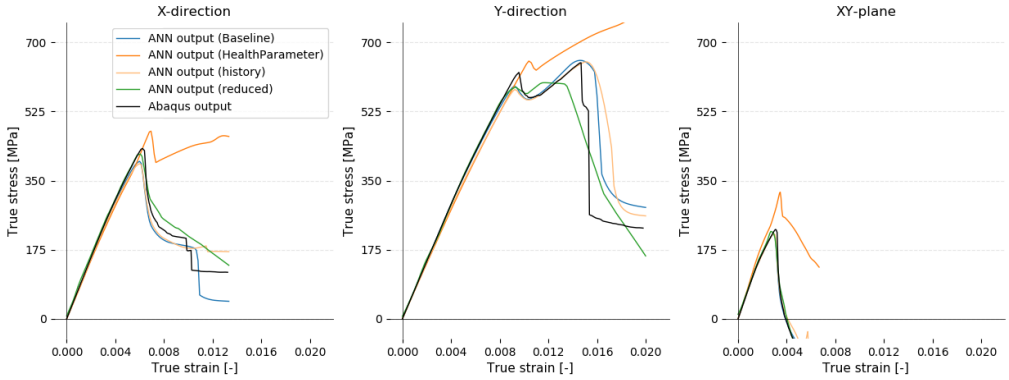
Minimum input design For this ANN configuration, a surprisingly good fit is found in combination with a short training time and relatively compact network architecture: even with a HL configuration of $N_{1,2} = (40, 30)$, a good fit can be obtained. The only imperfection about the ANN response is in the axial load prediction shown in Figure 5.9b. It seems that this ANN configuration is not able to capture the repeated load peaks properly, whereas the curves with one clearly defined peak are predicted with excellent accuracy. While the 'minimum input' network provides a good fit at low computational cost, it has some disadvantages as explained in Subsection 3.2.4. The final choice of hyperparameters for this configuration ANN is the following.

- $\alpha = 0.008$, $\gamma = 0.004$, $\beta_1 = 0.95$, $\beta_2 = 0.999$
- $N_{epochs} = 3,000$
- $N_{1,2} = (40, 30)$
- Initialiser = *Lecun uniform*

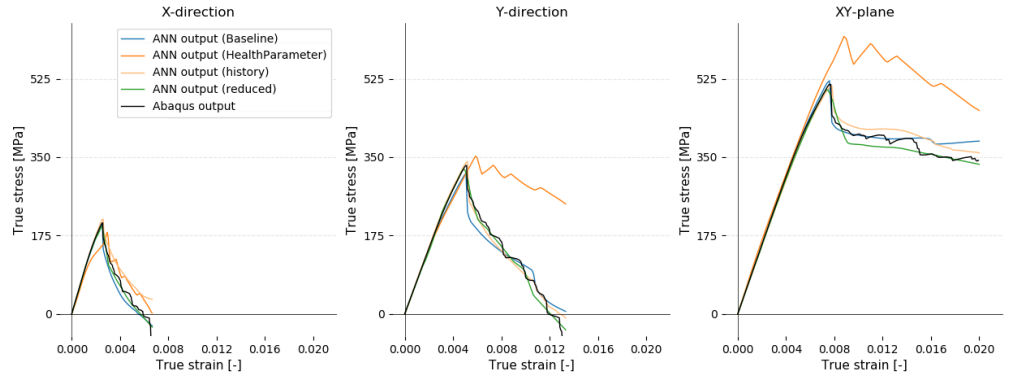
5.2.4. Performance and Discussion

The composite plate with an elliptic cutout considered in this section featured complex mechanical behaviour resulting from both material and geometrical aspects. The combination of a cutout and damage initiation and progression produces a complex mechanical response and proved to be a challenging case to model with an ANN. In fact, the majority of the process described in this section was first carried out on a simplified biaxial load case before adding shear. This was useful to gain a better understanding of the influence of hyperparameters on the response, after which the complete loading was considered.

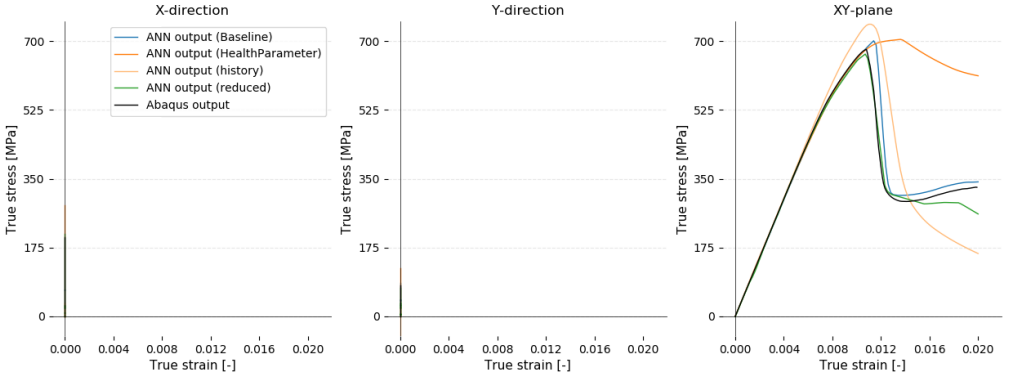
First of all, the data generator module executed a series of 37 simulations resulting from a grid of 2 axial loads in the x and y direction, as well as a shear load in the xy plane. Including the variation in load increments, a total of 148 data files formed the training data set for the ANN. It



(a) Loading ratio $\epsilon_x/\epsilon_y/\epsilon_{xy} = 13.3/20.0/6.67$



(b) Loading ratio $\epsilon_x/\epsilon_y/\epsilon_{xy} = 6.67/13.3/20.0$



(c) Loading ratio $\epsilon_x/\epsilon_y/\epsilon_{xy} = 0.0/0.0/20.0$

Figure 5.9: Comparison of ANN responses for several input and output configurations for the composite plate model, on the three chosen verification cases

was found that the amount of simulations could be drastically reduced by only considering points in the load grid that contained at least one maximum load. This approach did not result in a decreased capability of the ANN to capture the mechanical response.

The next step was to initialise and optimise the ANN hyperparameters to capture the composite plate's mechanical behaviour. To do so, an appropriate architecture was first determined based on a visual comparison between the mechanical responses, as this allows for a clear indication of how well the ANN can capture the non-linearities. It was shown that a 2-HL network with 60 and 40 nodes in each respective HL can capture the complexity of the composite plate model under

in-plane loading. After achieving this, other hyperparameters such as the training time, initialiser and optimiser settings were tuned to specifically suited for the current problem.

Apart from a baseline ANN which took the current stress-strain state as well as the strain increment as input and the stress increment as output, other configurations were considered and evaluated. A network with a set of health parameters as additional inputs, which provide a measure of the stiffness reduction due to plasticity and damage, was shown to perform significantly worse than the baseline ANN design even with longer training times. A measure that did boast some improvement was shown to be the inclusion of a historic stress-strain state in the input vector. While the improvement in the actual mechanical response is marginal, the ANN does train about twice as fast compared to the baseline ANN. Finally, an ANN which only takes the updated strain state as input and predicts the updated stress state directly as output is considered. While this configuration converges after a short training time and with a slightly reduced network architecture, it provides an excellent fit for almost each loading configuration. The disadvantages of this set-up have to be kept in mind though, as it is not possible to employ this architecture when generalising to cyclic or dynamic loading and is limited in using the tangent material stiffness matrix.

The final performance of the framework for the composite plate with an elliptic hole is shown in Table 5.6. While the simulations are by far the most costly part, the final result provides a clear view of why the current approach may be beneficial. After the initial investment in generating data and an ANN training time of less than 10 minutes, an effective and accurate homogenised model is obtained. This predictor features a computational cost of computing the chosen substructure's mechanical response that is two orders of magnitude lower than a conventional FEM analysis.

Table 5.6: Computational time of each module in the framework for the CFRP plate model

Action	time [s]
Data set generation (37 analyses)	43,000
ANN training	550
ANN-based FE model	3

5.3. Summary

The current chapter presented the implementation of the framework proposed in Chapter 3. For each of the two Finite Element (FE) models outlined in Chapter 4, a design of experiments was performed to specify the dimensions and loading combinations to be present in the data set. Subsequently, an Artificial Neural Network (ANN) was set up and trained on the generated data by choosing the architecture, training time, initialiser and optimisation parameters. Its performance was verified through a prediction of verification cases which were withheld from the training data.

Firstly, the high-strength steel plate was considered in order to gain an understanding of how to set up the design of experiments such that a representative and sufficiently voluminous data set could be established. This FE model, which essentially represents an elasto-plastic constitutive model without further complexities, proved to be particularly useful to efficiently build a good initial ANN configuration. For the data set, it was found that a grid of loading configurations with 4 linearly spaced steps between the minimum and maximum loads in each direction resulted in a sufficiently good ANN response. Furthermore, a network architecture of two Hidden Layers (HL) and 10 nodes per HL was found to be a good configuration with sufficient complexity to capture the mechanical response under biaxial tension. A training time of 1,500 epochs was chosen in combination with the *lecun_uniform* initialiser and *Nadam* optimiser. This resulted in an excellent verification response with a maximum error in the predicted stress of less than 5%.

A less trivial demonstration of the framework's capabilities was performed on a laminated composite plate model, loaded in biaxial tension and shear. It featured both a geometrical irregularity in the form of an elliptic cut-out and a material model with Hashin damage, thus introducing a complexity on two fronts. Departing from the findings of the steel plate model, it was found that a load grid of 4 linearly spaced steps between the minimum and maximum load was still sufficient to train the ANN. In fact, it was found that the amount of simulations could be reduced by only considering points in the load grid which contain at least one maximum load, resulting in a total of 37 FE analyses. Similar to the steel model, it was found that a 2-HL configuration is suitable for the mechanical response model and additional layers do not improve the response. Furthermore, an architecture of 60 and 40 nodes in each HL respectively was found to provide the best fit in combination with a training time of 10,000 epochs. The choice of initialiser and optimiser were re-evaluated and the decision for the *lecun_uniform* initialiser and *Nadam* optimiser was reinforced. Similar to the steel plate model, the maximum observed verification error of the ANN constitutive model was below 5%.

Apart from the baseline ANN set-up with the current stress-strain state and strain increment as input and the stress increment as output, three other configurations were considered. The same design procedure as before was followed to optimise them and their performance was compared again with three verification cases. Interestingly, the ANN configuration with health parameters performed significantly worse than the baseline and did not seem able to capture the response of the damaged region properly. Furthermore, it was observed to be somewhat sensitive to overfitting from the training and testing accuracy. In contrast, the ANN with history inputs did perform roughly the same as the baseline at half the training time. The final configuration, which only took the strain state as input and the stress state as output performed slightly worse than the baseline. Still, it was found to converge with an architecture which was 33% smaller than the baseline ANN and 70% less training time.

Apart from demonstrating the effectivity of the ANN to capture the homogenised mechanical response of both an elasto-plastic material model and a composite plate with a cut-out, the performance of the methodology was evaluated. For both models, the final result was a fully explicit constitutive model of the homogenised substructure response with almost no computational cost: any loading combination could be evaluated in 3 seconds. This resulted a reduction of one order of magnitude for the relatively simple steel plate and two orders of magnitude in case of the more complex composite plate, as compared to a regular FE analysis.

6

Conclusions

The driving objective of this master's thesis was to improve the accuracy and/or computational efficiency of substructure homogenisation in Finite Element Method (FEM) modelling by implementing a constitutive model based on an Artificial Neural Network (ANN), to be trained on a detailed and representative data set. As motivated in the introduction, the current approach to homogenised substructuring in FEM is limited by several aspects. Especially the complexity of the resulting material model, which is to represent the homogenised behaviour of the substructured section, is often confined to the linear-elastic regime.

The work presented in this thesis shaped the foundation of a novel framework that allows the creation of a homogenised substructure representation that is simultaneously efficient, accurate and capable of capturing complexities resulting from structural or material irregularities. In order to draw conclusions, the research questions stated in the introduction are answered. However, before answering the research questions, the constructed procedure is outlined.

The framework to modelling substructural behaviour constructed for this work consisted of three parts: the data generator, the ANN set-up and optimisation and finally, its implementation in FEM.

- The *data generator module* consists of a script that performs several actions within the FEM software package Abaqus. Before employing the data generator, a suitable region to be modelled as a substructure is to be identified. Then, the substructure is isolated and implemented in the script as a parameterised model which contains at least the loading conditions as parameters. Furthermore, it was pointed out that one could include other features (e.g. a cut-out size) as a variable to broaden the ANNs domain. Once the FEM model is constructed, the data generator module is capable of automatically generating it for each configuration, applying boundary conditions, defining the loads, running the analysis and finally, extracting and writing the homogenised response to a data set. The result is a series of data obtained from all FEM simulations that were specified by the user in the Design of Experiments (DoE) configuration (including loading conditions, increments and potentially other parameters).
- The *ANN training module*'s inputs are the data from the data generator module and a series of hyperparameters, which are the global user-specified parameters that define the ANN set-up and training process. In order to prepare the data for the ANN training, it was found that normalisation has to be applied to facilitate an effective training process. Furthermore, the data is split into a training and testing data set such that the ANN performance can be verified on independent data. After defining the network's architecture and setting the other hyperparameters, the training process can be commenced and afterwards, its accuracy evaluated.

- The third and final module of the proposed procedure covers the *integration in Abaqus*. This can be achieved by constructing a customised material model for a suitable element type in Abaqus, which can then be readily used in a finite element model.

Now that the framework is constructed, let us get back to answering the research questions.

I What artificial neural network set-ups, architectures and training procedures are suitable to model constitutive behaviour? Furthermore, how can a sufficiently representative training data set be constructed from a minimum amount of simulations?

After the construction of the framework, it was possible to investigate the requirements of the data set. It was found that, in order to form a representative data set of mechanical behaviour, each independent loading variable requires at least four linearly spaced steps between its minimum and maximum. In other words, when two loading variables are considered in a biaxial load case, a *loading grid* is constructed of $4^2 - 1 = 15$ entries which contains all possible combinations except for the trivial all-zero load case. Similarly, a grid with biaxial tension and in-plane shear loading contains $4^3 - 1 = 63$ entries. As each point on the grid represents a full FE simulation with potentially high computational expense, it was investigated if the points on the grid could be reduced. Indeed, it was found that the data set was still sufficiently representative for the ANN to capture the mechanical behaviour after removing all points that did not contain at least one maximum load. This reduced the amount of required simulations to $4^3 - 3^3 = 37$.

Concerning the ANN architecture, it was found that a *sequential, densely connected lay-out* is suitable to capture the complexities of mechanical behaviour. More specifically, a 2 Hidden Layer (HL) architecture was found to be ideal. The amount of nodes (artificial neurons) per HL have to be optimised based on the amount of independent loading conditions and other input variables, as well as the complexity of the substructure to be modelled (e.g. damage propagation, failure and cut-outs). To capture a homogeneous elasto-plastic steel material model in biaxial tension, each HL requires 10 nodes. For a composite plate with an elliptic cut-out and Hashin damage under biaxial tension and in-plane shear, the two HLs require around 60 and 40 nodes respectively.

Apart from the HLs, one may choose different input/output vector configurations. The current work has investigated the performance of several alternatives for the *baseline* configuration. The baseline input vector features the current stress and strain state as well as the current strain increment, whereas the output vector predicts the corresponding stress increment. An alternative input vector was proposed that contained a set of *health parameters* to provide the network with a measure of the current state of damage. Interestingly, it was found that this configuration performed considerably worse than the baseline. Some improvement was found for another configuration, that contained a *historic stress-strain state* (of the previous increment) as additional input. While the response was approximately the same, the convergence of the training process was improved with a factor of 2. Finally, a *minimum input* configuration was designed which featured only the strain state as input and the stress state as output. The convergence of this ANN was found to be the fastest with less nodes per HL, while a minor reduction in accuracy was observed.

Finally, the leftover hyperparameters were determined. Among these were the optimisation algorithm, optimisation settings and initialisers. After comparing the performance of the ANN with several configurations, it was concluded that the best optimiser is the Adaptive Gradient Descent with Nesterov-Accelerated Momentum (*Nadam*) scheme. The most important optimisation parameters are the learning rate and decay rate, which were both found to be optimal when roughly equal to 0.008. Furthermore, the best performing initialisation method was found to be the *Lecun* scheme.

II To what extent can an artificial neural network be used to capture complex homogenised behaviour of a substructure? Furthermore, what are the conditions such that a substructure is fit for usage with the proposed method?

Within the scope of the current work, two benchmark cases of increasing complexity were investigated. Firstly, the ANN was employed to capture biaxial loading on an elasto-plastic constitutive

model of a high-strength steel without additional material and structural complexities. Secondly, a composite plate with Hashin damage and an elliptical cut-out under biaxial tension and in-plane shear was investigated. Based on the results presented in this work, it can be concluded that an ANN-based constitutive model is *fully capable* of capturing the homogenised behaviour of both models. This statement is valid for a substructure which features (almost) uniform stresses and strains at its boundary and thus, a *linear displacement and force distribution*.

III Is it feasible to integrate an artificial neural network into a finite element analysis procedure as a constitutive model?

After training an ANN which fits the homogenised constitutive behaviour of the substructure sufficiently well, it was found that its integration in Abaqus can be performed by utilising a *User Material Subroutine* (UMAT). Using UMAT allows the integration of a fully customised material model for chosen elements in Abaqus. Because the inputs and outputs of the ANN and UMAT are similar and the fact that an ANN is an explicit equation (albeit potentially a large one), the UMAT model can be readily constructed and implemented. It was found, however, that a particular requirement for this UMAT is the construction of a material stiffness matrix. In contrast to a conventional constitutive model, the stiffness matrix is not used in the ANN-based constitutive model. Still, it is required to compute this quantity for the element and global level FEM calculations and the current work formulated three methods to do just that. Thus, it can be concluded that it is indeed possible to integrate an ANN into a FEM procedure as a fully functional constitutive model.

IV If the previous question is answered positively, what are the gains in computational time and/or accuracy from using the artificial neural network-based set-up?

The methodology proposed in this work provides the user with the possibility to model the complex homogenised behaviour of substructures. In doing so, a trade-off between accuracy and efficiency exists. On one hand, the capabilities of the ANN-based constitutive model allow for an improvement in accuracy as one can predict complex mechanical responses. The computational cost here lies in generating the training data, which depend on high-fidelity FEM simulations. The generation of data for the composite plate including damage with an elliptical cut-out in biaxial tension and in-plane shear required half a day of computation time, for example. This number may dramatically increase for more complex cases with more independent loading conditions. However, once the data is obtained, it takes less than 10 minutes to train the ANN and one can fully reconstruct any (homogenised) mechanical response within the training domain in 3 seconds. For the composite plate, it can therefore be concluded that the gain in computational time is over two orders of magnitude with a similar accuracy. Compared to traditional homogenised substructural models, it is pointed out that it takes more time to establish the ANN-based model as the former method does not require the data generating phase. Still, as pointed out before, the ANN-based model offers a major increase in the domain and accuracy compared to traditional homogenised models.

7

Future Work

The present work paved the first step of a new way of looking at Finite Element Method (FEM) modelling. While the combination of design of experiments, computational analyses and data extraction is not new, its application to obtaining the homogenised substructural response and its direct integration in FEM has not been demonstrated before.
This part of the report elaborates on how to improve and continue the presented framework, based on the author's perspective and opinions. Based on the already proposed and described methodology, the following areas of potential improvement are identified.

- **Buckling:** This work only considers (biaxial) tensile loads and in-plane shear, with the specific assumption that no buckling occurs in shear. The reason was the extra complexity and simulation cost that buckling would introduce, which was outside the scope of this work. Still, it is expected that the effects of buckling can be readily captured by an Artificial Neural Network (ANN) by slightly enlarging the Hidden Layer (HL) sizes.
- **Generalise to 3D:** The code written for this thesis is largely generalised already to work with three-dimensional models. Again, due to the computational cost, the established framework is not yet applied for such a case. A critical view has to be adopted towards such simulations though, as the generalised loading on a 3D structure consists of 6 independent entries. This exponentially increases the required amount of simulations to generate a data set, which may cause the ANN-based procedure to be too expensive for such a generalised case.
- **Stiffness matrix verification:** While three different methods are proposed in the current work to construct a stiffness matrix, it was found that Abaqus occasionally has difficulties in employing it effectively. A deeper understanding of the construction and definition of this quantity should be obtained in order to verify the correct implementation.
- **Error estimation:** The ANNs employed in this work use a simple mean absolute error cost function. While this seems to be a suitable choice since an excellent fit can be obtained, it is recommended to investigate a customised error function. The reason is that the current baseline ANN design takes an incremental approach (it predicts stress increments), which leads to the accumulation of errors. This may result in a high theoretical accuracy when in fact, a small error at the beginning of the mechanical response may have a disastrous effect later in the response. Therefore the implementation of a custom error function, that evaluates the incremental accuracy instead of the accuracy of each predefined data point, may improve the training process and accuracy even more.
- **Cross-validation:** When a small amount of (expensive) data is available, it is often not desired to sacrifice a significant chunk of it to form the testing data set as it may enlarge the bias. The importance of test data is, however, undisputed as it is an invaluable indicator of

over-fitting and a means of verifying the capability of the ANN to generalise on the trends in the test data. The method employed in this work considers a fixed set of training data, called 'holdout cross-validation'. An improvement when one prefers to use all available data for training but still verify the model may be found in 'k-fold cross-validation'. This scheme forms k chunks of data and rotates their roles as test data when training, averaging their errors. Similar techniques exist such as 'Stratified K-Fold Cross Validation' and 'Leave-P-Out Cross Validation', each boasting their own benefits.

- **Hashin mesh dependencies:** The mesh convergence analysis for the composite plate model led to an adequate mesh choice within this thesis' scope. Still, a clear mesh dependency was still observed especially for the mechanical response region where damage propagation occurs. It is recommended to employ a more advanced damage model to obtain a smoother and more consistent response which may improve the ANN response as well.

The proposed framework offers plenty of opportunities for future work. The main thoughts on what steps may be taken next are proposed below.

- **Validation:** A major addition to the developed methodology would be to perform a validation by scaling up the approach to a real-world problem. More specifically, the application of the framework to an aircraft's wing box or fuselage would provide a clear indication of its advantages and limitations. In such a structure, it may be desirable for example to capture the area around the lightening holes or window holes to take advantage of repeatability.
- **User element subroutine:** Ultimately, the current author believes that the material-based model using UMAT has to be generalised to an element-based approach using UEL: a User-Element. This will provide the opportunity to capture not only axial deformation and shearing, but also bending and torsion movements. Furthermore, the use of numerical integration can be omitted entirely at element and material level by directly predicting the element response. Another advantage is the possibility to capture a higher-order boundary response.
- **Experimental data:** Since ANNs can be trained on any type of data (as long as it is sufficiently representative) and are known for their noise resistance, they may be trained on experimental data as well. Not all materials are (fully) captured yet in a constitutive model, indicating a potential area of application. Therefore it is recommended to explore the possibilities and benefits of capturing experimental material behaviour.
- **Dynamic behaviour:** In the current work, it has not been investigated if the ANN may be suitable for modelling dynamic behaviour. A recommendation is to investigate which particular ANN input and output configuration is suitable for capturing this behaviour and if the generation of data is feasible.

Bibliography

[1] C. A. Felippa, "Superelements and global-local analysis," in *Introduction to Finite Element Methods*. Boulder, Colorado, USA: University of Colorado, 2004, ch. 10.

[2] M. G. Ostergaard, A. R. Ibbotson, O. le Roux, and A. M. Prior, "Virtual testing of aircraft structures," *CEAS Aeronautical Journal*, vol. 1, no. 1, pp. 83–103, 2011.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] T. M. Mitchell, *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[5] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[6] A. Ng, "Neural networks and deep learning," Date 2018.

[7] X. Wu and J. Ghaboussi, "Neural network-based material modeling," Thesis, University of Illinois, 1991.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[9] D. O. Hebb, *Organization of Behavior*. New York: Wiley, 1949.

[10] F. Rosenblatt, *Principles of Neurodynamics. Preceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Lab inc Buffalo NY, 1961.

[11] Tensorflow, "A neural network playground," https://playground.tensorflow.org, 2016, accessed 12-03-2018.

[12] A. I. Galushkin, *Neural Networks Theory*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2007.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, p. 533, 1986.

[14] A. Ng, "Improving deep neural networks: Hyperparameter tuning, regularization and optimization," Date 2018.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[16] D. Sussillo, "Random walks: Training very deep nonlinear feed-forward networks with smart initialization," *CoRR*, vol. abs/1412.6558, 2014.

[17] P. Y. Papalambros and D. J. Wilde, *Principles of optimal design: modeling and computation*. Cambridge university press, 2000.

[18] E. F. M. Moreira, "Neural networks with adaptive learning rate and momentum terms," Institut Dalle Molle D'Intelligence Artificielle Perceptive, Report, 1995.

[19] P. E. Rauber, S. G. Fadel, A. X. Falcão, and A. C. Telea, "Visualizing the hidden activity of artificial neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 101–110, Jan 2017.

[20] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in neural information processing systems*, 1990, pp. 524–532.

[21] T. Nakama, "Comparisons of single- and multiple-hidden-layer neural networks," in *Advances in Neural Networks – ISNN 2011*, D. Liu, H. Zhang, M. Polycarpou, C. Alippi, and H. He, Eds. Berlin, Heidelberg: Springer, 2011, pp. 270–279.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[23] C. Bisagni and L. Lanzi, "Post-buckling optimisation of composite stiffened panels using neural networks," *Composite Structures*, vol. 58, no. 2, pp. 237–247, 2002.

[24] W. Ruijter, R. Spallino, L. Warnet, and A. de Boer, "Optimization of composite panels using neural networks and genetic algorithms," in *Second MIT Conference on Compurational Fluid and Solid Mechanics*, K. J. Bathe, Ed. Elsevier Science Ltd, 2003, Conference Proceedings, pp. 2359–2363.

[25] P. Hajela and L. Berke, "Neural networks in structural analysis and design: An overview," *Computing Systems in Engineering*, vol. 3, no. 1, pp. 525–538, 1992.

[26] L. Lanzi, C. Bisagni, and S. Ricci, "Neural network systems to reproduce crash behavior of structural components," *Computers & Structures*, vol. 82, no. 1, pp. 93–108, 2004.

[27] M. Bessa and S. Pellegrino, "Design of ultra-thin shell structures in the stochastic post-buckling range using bayesian machine learning and optimization," *International Journal of Solids and Structures*, vol. 139-140, pp. 174 – 188, 2018.

[28] Z. Fu, J. Mo, L. Chen, and W. Chen, "Using genetic algorithm-back propagation neural network prediction and finite-element model simulation to optimize the process of multiple-step incremental air-bending forming of sheet metal," *Materials & Design*, vol. 31, no. 1, pp. 267–277, 2010.

[29] A. R. Shahani, S. Setayeshi, S. A. Nodamaie, M. A. Asadi, and S. Rezaie, "Prediction of influence parameters on the hot rolling process using finite element method and neural network," *Journal of Materials Processing Technology*, vol. 209, no. 4, pp. 1920–1935, 2009.

[30] R. I. Levin and N. A. J. Lieven, "Dynamic finite element model updating using neural networks," *Journal of Sound and Vibration*, vol. 210, no. 5, pp. 593–607, 1998.

[31] J. Ghaboussi, J. H. Garrett, and X. Wu, "Knowledge-based modeling of material behavior with neural networks," *Journal of Engineering Mechanics*, vol. 117, no. 1, pp. 132–153, 1991.

[32] Y. Hashash, S. Jung, and J. Ghaboussi, "Numerical implementation of a neural network based material model in finite element analysis," *International Journal for numerical methods in engineering*, vol. 59, no. 7, pp. 989–1005, 2004.

[33] A. A. Javadi, P. Tan, and M. Zhang, "Neural network for constitutive modelling in flnite element analysis," University of Exeter, Report, 2003.

[34] J. Ghaboussi, D. A. Pecknold, M. Zhang, and R. M. Haj-Ali, "Autoprogressive training of neural network constitutive models," *International Journal for Numerical Methods in Engineering*, vol. 42, no. 1, pp. 105–126, 1998.

[35] J. Ghaboussi and D. E. Sidarta, "New nested adaptive neural networks (nann) for constitutive modeling," *Computers and Geotechnics*, vol. 22, no. 1, pp. 29–52, 1998.

[36] Y. M. A. Hashash, H. Song, S. Jung, and J. Ghaboussi, "Extracting inelastic metal behaviour through inverse analysis: A shift in focus from material models to material behaviour," *Inverse Problems in Science and Engineering*, vol. 17, no. 1, pp. 35–50, 2009.

[37] G. J. Yun, J. Ghaboussi, and A. S. Elnashai, "A new neural network-based model for hysteretic behavior of materials," *International Journal for Numerical Methods in Engineering*, vol. 73, no. 4, pp. 447–469, 2008.

[38] G. J. Yun, J. Ghaboussi, and A. S. Elnashai, "Self-learning simulation method for inverse nonlinear modeling of cyclic behavior of connections," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 33, pp. 2836–2857, 2008.

[39] Y. Teboub and P. Hajela, "A neural network based damage analysis of smart composite beams," in *4th Symposium on Multidisciplinary Analysis and Optimization*, ser. Multidisciplinary Analysis Optimization Conferences. Troy, NY, U.S.A.: American Institute of Aeronautics and Astronautics, 1992, Conference Proceedings, p. 4685.

[40] J. N. Kudva, N. Munir, and P. W. Tan, "Damage detection in smart structures using neural networks and finite-element analyses," *Smart Materials and Structures*, vol. 1, no. 2, p. 108, 1992.

[41] R. M. V. Pidaparti and M. J. Palakal, "Material model for composites using neural networks," *AIAA Journal*, vol. 31, no. 8, pp. 1533–1535, 1993.

[42] P. Labossiere and N. Turkkan, "Failure prediction of fibre-reinforced materials with neural networks," *Reinforced Plastics and Composites*, vol. 12, no. 12, pp. 1270–1280, 1993.

[43] C. S. Lee, W. Hwang, H. C. Park, and K. S. Han, "Failure of carbon/epoxy composite tubes under combined axial and torsional loading 1. experimental results and prediction of biaxial strength by the use of neural networks," *Composites Science and Technology*, vol. 59, no. 12, pp. 1779–1788, 1999.

[44] P. A. M. Lopes, H. M. Gomes, and A. M. Awruch, "Reliability analysis of laminated composite structures using finite elements and neural networks," *Composite Structures*, vol. 92, no. 7, pp. 1603–1613, 2010.

[45] H. Man and G. Prusty, "Neural network modelling for damage behaviour of composites using full-field strain measurements," *Composite Structures*, vol. 93, no. 2, pp. 383–391, 2011/01/01/ 2011.

[46] T. Kirchdoerfer and M. Ortiz, "Data-driven computational mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 304, pp. 81 – 101, 2016.

[47] L. T. K. Nguyen and M.-A. Keip, "A data-driven approach to nonlinear elasticity," *Computers & Structures*, vol. 194, pp. 97 – 115, 2018.

[48] F. Chinesta, P. Ladeveze, R. Ibanez, J. V. Aguado, E. Abisset-Chavanne, and E. Cueto, "Data-driven computational plasticity," in *International Conference on the Technology of Plasticity*, ser. Procedia Engineering, vol. 207. Elsevier, 17-09-2017 2017, Conference Proceedings, pp. 209–2014.

[49] Z. Liu, M. Bessa, and W. K. Liu, "Self-consistent clustering analysis: An efficient multi-scale scheme for inelastic heterogeneous materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 306, pp. 319 – 341, 2016.

[50] M. Bessa, R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen, and W. K. Liu, "A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality," *Computer Methods in Applied Mechanics and Engineering*, vol. 320, pp. 633 – 667, 2017.

[51] F. Fritzen and O. Kunc, "Two-stage data-driven homogenization for nonlinear solids using a reduced order model," *European Journal of Mechanics - A/Solids*, vol. 69, pp. 201 – 220, 2018.

[52] A. Oishi and G. Yagawa, "Computational mechanics enhanced by deep learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 327, pp. 327 – 351, 2017, advances in Computational Mechanics and Scientific Computation—the Cutting Edge.

[53] Y. Chen, L. Dong, B. Wang, Y. Chen, Z. Qiu, and Z. Guo, "A substructure-based homogenization approach for systems with periodic microstructures of comparable sizes," *Composite Structures*, vol. 169, pp. 97 – 104, 2017, in Honor of Prof. Leissa.

[54] iMechanica, "Writing user subroutines with abaqus," http://imechanica.org/files/Writing%20User%20Subroutines%20with%20ABAQUS.pdf, 2012, accessed on 20-06-2018.

[55] O. Zienkiewicz, *The finite element method in engineering science*. McGraw-Hill, 1971.

[56] B. Chen, "Non-linear modelling lecture slides week 1," https://brightspace.tudelft.nl/, 2018, accessed on 01-11-2018.

[57] Mathworks, "Matlab deep learning toolbox," https://www.mathworks.com/, 1992, commercial Deep Learning Library for Matlab.

[58] Wolfram Research, "Wolfram mathematica," https://www.wolfram.com/mathematica, 1988, commercial Deep Learning Library.

[59] Google Brain Team, "Tensorflow," www.tensorflow.org, 2015, open-Source Data Flow Library for Python.

[60] U. Montreal Institute for Learning Algorithms, "Theano," https://github.com/Theano/Theano, 2007, open-Source Deep Learning Library for Python.

[61] Various Authors, "Keras," www.keras.io, 2015, open-Source Deep Learning Library.

[62] Dassault Systèmes, "Abaqus 2016 documentation: Analysis user's manual," http://abaqus.software.polimi.it/v6.17/index.html, 2016, accessed on 12-09-2018.

[63] N.-C. Fahlbush, "Entwicklung und analyse mikromechanischer modelle zur beschreibung des effektivverhaltens von geschlossenzelligen polymerschäumen," dissertation, TU Darmstadt, 2015.

[64] S. Li and A. Wongsto, "Unit cells for micromechanical analysis of particle-reinforced composites," *Mechanics of Materials*, vol. 36, pp. 543–572, 2004.

[65] W. C. Young and R. G. Budynas, *Roark's formulas for stress and strain*. McGraw-Hill, 2002, vol. 7.

[66] I. Daniel and O. Ishai, *Engineering Mechanics of Composite Materials*, ser. Engineering mechanics of composite materials. Oxford University Press, 2006, no. v. 13. [Online]. Available: https://books.google.nl/books?id=x5S_QgAACAAJ

[67] B. Chen, "Numerical modelling of scale-dependent damage and failure of composites," Thesis, National University of Singapore, 2013.

[68] Q. Yang and B. Cox, "Cohesive models for damage evolution in laminated composites," *International Journal of Fracture*, vol. 133, no. 2, pp. 107–137, 2005.

[69] P. P. Camanho, "Failure criteria for fibre-reinforced polymer composites," Universidade do Porto, Report, 2002.

[70] Z. Hashin, "Failure criteria for unidirectional fiber composites," *Journal of Applied Mechanics*, vol. 47, no. 2, pp. 329–334, 1980.

[71] C. Kassapoglou, *Design and Analysis of Composite Structures : With Applications to Aerospace Structures*, 2nd ed., ser. Wiley Aerospace Series. New York: Wiley, 2013.

[72] B. Chen, T. Tay, P. Baiz, and S. Pinho, "Numerical analysis of size effects on open-hole tensile composite laminates," *Composites Part A: Applied Science and Manufacturing*, vol. 47, pp. 52 – 62, 2013.

A

User Material Subroutine in Fortran

The current appendix contains the Fortran code corresponding to Subsection 3.4.2. It can be used with Abaqus, provided that it is connected to a Fortran compiler that supports the free-form format.

The presented code is an example for a 2D constitutive model with a [9-80-60-3] architecture and activation functions equal to ReLU and tanh in the Hidden Layers (HL) and output layer respectively. Due to the large size of the weight and bias vector definitions, these expressions have been shortened.

```
1  !**************************************!
2  !    Abaqus UMAT neural network interface !
3  !            2D shell element             !
4  !    created by Tom Gulikers, TU Delft    !
5  !                                         !
6  !**************************************!
7  ! enforce the use of free-form Fortran
8   !DIR$ FREEFORM
9
10 !------ include external modules ------------------------
11  include 'globals/parameter_module.f90'
12 !-------------------------------------------------------
13
14 SUBROUTINE UMAT(STRESS, STATEV, DDSDDE, SSE, SPD, SCD, RPL,&
15  &DDSDDT, DRPLDE, DRPLDT, STRAN, DSTRAN, TIME, DTIME, TEMP, DTEMP,&
16  &PREDEF, DPRED, CMNAME, NDI, NSHR, NTENS, NSTATV, PROPS, NPROPS,&
17  &COORDS, DROT, PNEWDT, CELENT, DFGRD0, DFGRD1, NOEL, NPT, LAYER,&
18  &KSPT, KSTEP, KINC)
19
20    ! load FNM modules
21    use parameter_module,          only: NDIM, DP, ZERO, ONE, SMALLNUM
22
23    include 'aba_param.inc'
24
25    CHARACTER(len=8) :: CMNAME
26
27    ! assign dimension to subroutine variables
28    DIMENSION STRESS(NTENS), STATEV(NSTATV), DDSDDE(NTENS, NTENS), &
29    &DDSDDT(NTENS), DRPLDE(NTENS), STRAN(NTENS), DSTRAN(NTENS),&
30    &PREDEF(1), DPRED(1), PROPS(NPROPS), COORDS(3), DROT(3, 3),&
```

```
31    &DFGRD0(3, 3), DFGRD1(3, 3)
32
33    ! initialise fixed-size variables
34    integer                                    :: i, j, k, l, NB, NC, n_dim
35    real(DP)                                   :: S_sigma, S_epsilon, S_dsigma, S_depsilon
36    real(DP), dimension (3)                    :: sigma
37    real(DP), dimension (3, 1)                 :: sigma_NN_i1, sigma_out, C, b_final
38    real(DP), dimension (3, 3)                 :: Dmat
39    real(DP), dimension (80, 1)                :: B, b_BA
40    real(DP), dimension (9, 1)                 :: network_input
41    real(DP), dimension (80, 9)                :: w_BA
42    real(DP), dimension (3, 60)                :: w_final
43
44    ! initialise allocatable variables for ANN calculations
45    real(DP), dimension (:,:), allocatable     :: w_CB, b_CB, w_DC, b_DC, w_ED, b_ED
46    real(DP), dimension (:,:), allocatable     :: D, E, x, net_input, output
47
48    ! fixed variable assignment
49    CMNAME = "compBrick_2_2_2"
50    n_dim = 3
51    S_sigma = 1000.0
52    S_epsilon = 0.021
53    S_dsigma = 50.
54    S_depsilon = 0.005
55    NB = 80
56    NC = 60
57
58
59    ! ------------------------
60    ! Create ANN input vector
61    ! ------------------------
62
63    do i = 1, n_dim
64      network_input(i,1) = STRESS(i)/S_sigma
65      network_input(i+n_dim,1) = STRAN(i)/S_epsilon
66      network_input(i+n_dim*2,1) = DSTRAN(i)/0.001
67    end do
68
69
70    ! -------------------------------
71    ! Weight and bias definitions
72    ! -------------------------------
73
74    ! define weights of HL1
75    w_BA(1,:) = (/-0.5271,-0.36723,...,0.01975,0.05472/)
76    w_BA(2,:) = (/-0.09239,-0.10259,...,-0.08813,-0.04219/)
77       :
78    w_BA(79,:) = (/-0.54296,0.13494,...,0.2957,0.2869/)
79    w_BA(80,:) = (/0.01263,-0.047,...,-0.02572,-0.07342/)
80
81    ! define biases of HL1
82    b_BA(:,1) = (/0.16489,-0.05282,...,0.18884,-0.08705/)
83
84
85    ! define weights of HL_2
86    allocate ( w_CB(60,80) )
87    w_CB(1,:) = (/-0.03796,-0.01735,...,-0.07002,-0.05267/)
88    w_CB(2,:) = (/-0.08553,0.08835,...,0.49728,0.09246/)
```

```
89      ⋮
90    w_CB(59,:) = (/-6.01257e-01,-8.90269e-03,··· ,1.28529e-01,-7.28613e-02/)
91    w_CB(60,:) = (/-0.01929,-0.08428,··· ,0.0117,-0.02988/)
92
93    ! define biases of HL_2
94    allocate ( b_CB(60,1) )
95    b_CB(:, 1) = (/-0.06229,0.08949,··· ,0.12228,0.03022/)
96
97
98    ! define weights of output layer
99    w_final(1,:) = (/0.01231,0.17144,··· ,-0.15706,0.04677/)
100   w_final(2,:) = (/-0.02736,0.0699,··· ,0.2984,-0.13267/)
101   w_final(3,:) = (/0.02906,0.07102,··· ,-0.23502,0.1402/)
102
103   ! define bias of output layer
104   b_final(:,1) = (/-0.02767,-0.00781,-0.02876/)
105
106
107   ! ------------------------------
108   ! Neural network calculations
109   ! ------------------------------
110
111   ! HL_1 assignment of variables
112   allocate ( x (9, 1))          ! neuron input
113   allocate ( net_input(80, 1)) ! shape same as w_BA.shape[0]
114   allocate ( output(80, 1))     ! shape same as w_BA.shape[0]
115
116   ! HL_1 calculation
117   x = network_input
118   net_input = matmul(w_BA,x) + b_BA
119   output = max(net_input,ZERO)    ! ReLu function
120   B = output
121   deallocate ( net_input)
122   deallocate (x)
123
124   ! HL_2 assignment of variables
125   allocate ( x(80, 1))           ! shape same as w_CB.shape[1]
126   allocate ( net_input(60, 1))   ! shape same as w_CB.shape[0]
127   x = output
128   deallocate (output)
129   allocate ( output(60, 1))      ! shape same as w_CB.shape[0]
130
131   ! HL_2 calculation
132   net_input = matmul(w_CB, x) + b_CB
133   output = max(net_input,ZERO)     ! ReLu function
134   C = output
135   deallocate (net_input)
136   deallocate (x)
137
138   ! output layer assignment of variables
139   allocate ( x(60, 1))              ! shape same as w_final.shape[0]
140   allocate ( net_input(3, 1))      ! shape same as w_final.shape[1]
141   x = output
142   deallocate (output)
143   allocate ( output(3, 1))         ! shape same as w_final.shape[1]
144
145   ! output layer calculation
146   net_input = matmul(w_final, x) + b_final
```

```
147     output = tanh(net_input)
148     sigma_NN_i1 = output
149
150     ! deallocate all temporary variables
151     deallocate (output)
152     deallocate (net_input)
153     deallocate (x)
154     deallocate (b_CB)
155     deallocate (w_CB)
156
157
158     ! ---------------------------------------
159     ! secant stiffness matrix calculation
160     ! ---------------------------------------
161     do i = 1, n_dim
162      do j = 1, n_dim
163       if (abs(stran(j)+DSTRAN(j)) < SMALLNUM) then
164        Dmat(i, j) = 0._dp
165       else
166        Dmat(i, j) = (stress(i) + sigma_NN_i1(i,1)*50.) / (stran(j)+DSTRAN(j))
167       end if
168      end do
169     end do
170
171
172     ! ---------------------------------------------------
173     ! Pass internal variables to subroutine outputs
174     ! ---------------------------------------------------
175      DDSDDE    = Dmat
176      STRESS(:)    = STRESS(:) + sigma_NN_i1(:,1)*S_dsigma
177
178 END SUBROUTINE UMAT
```