DELFT UNIVERSITY OF TECHNOLOGY

MASTER OF SCIENCE THESIS

The Neuromorphic Element: A Data-Driven Finite Element Formulation Using Self-Designing Neural Networks – Proof of Concept on Nonlinear Trusses

Author:
Nicolas Ruitenbeek
Student ID: 4291069

Supervisor:
Dr. Boyang Chen

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Aerospace Engineering at the Delft University of Technology.

Aerospace Structures and Computational Mechanics Research Group
Faculty of Aerospace Engineering, TU Delft

To be publicly defended on Thursday June 11, 2020, at 10:00AM
Delft, The Netherlands

Declaration of Authorship

I, Nicolas Ruitenbeek, declare that this thesis titled, "The Neuromorphic Element: A Data-Driven Finite Element Formulation Using Self-Designing Neural Networks – Proof of Concept on Nonlinear Trusses" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a Master of Science degree at the Delft University of Technology (TU Delft).
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- Sufficient information and resources are provided within this thesis to ensure reproducibility.

Signed: [illegible]

Date: 4 June, 2020

Graduation Committee

Chair holder:

Dr. Sergio Turteltaub
Aerospace Structures & Computational Mechanics - TU Delft

Committee members:

Dr. Boyang Chen
Aerospace Structures & Computational Mechanics - TU Delft

Prof. Clemens Dransfeld
Aerospace Manufacturing Technologies - TU Delft

Dr. Kunal Masania
Aerospace Manufacturing Technologies - TU Delft

Date:

This thesis is dedicated to my parents, Cindy and Jack, and my jaanu, Rhythm.

Thank you for your love, patience, and endless support. None of this would have been possible without you.

I love you.

"To understand God's thoughts one must study statistics."

- Florence Nightingale

Preface

I have written much of this thesis while under some type of lockdown during the Coronavirus pandemic. Most everybody during this period acknowledges the critical role that medical staff have played in providing all types of support during this emergency. I dedicate this thesis to those people, in particular the nurses as this year - 2020 - was some time ago designated as the International Year of the Nurse by the World Health Organisation [126]. 2020 was chosen because it marks the 200th anniversary of the birth of Florence Nightingale in Italy, which occurred in May of 1820. Her life was dedicated to nursing, and her work provided a model for modern nursing and modern policy making in that realm. What many do not realize is that her biographers call her the 'Passionate Statistician' as she treated the analysis and use of data as if statistics was her religion [88]. In fact, she has been credited with the invention of what we now call the pie chart. Her polar area diagram helped convince policy makers that more soldiers were dying from disease than from traditional battle injuries during the Crimean war. This tool provided an efficient and effective way to look at information through a different lens, giving new insights into what in those days was a complex problem. For this, and other advances, she was admitted as the first woman to the Royal Statistical Society. Since that period, in nursing and other professions, the interpretation and presentation of information has played a critical role. Whether during war or pandemic, speed and efficient use of resources are critical in winning a battle. A day or month can make a difference.

This thesis continues in the tradition of developing new tools that make the interpretation of engineering information more efficient. My own interest in developing such tools started during my studies in structural engineering, and in using traditional methods such as finite element analysis. I was also interested in forensic structural engineering and potential tools for damage prediction. It is perhaps a matter of impatience (waiting for computer run programmed scripts to completion) that caused me to muse about whether artificial intelligence (AI) might be part of that toolkit. I am of the generation that has grown up with so-called "smart" systems, where everything from vacuum cleaners to cars managed to do some of the thinking and decision-making for us. What started as curiosity turned into greater interest in doing theoretical or practical research in developing such tools. An internship at Airbus Defense and Space in 2018 opened my eyes to the potential practical applications of AI in the aerospace field. I thank my colleagues there, especially Lex Meijer, Henk Cruijssen, and Garmt Grommers for offering their mentoring during some design challenges where I was using more traditional analytical techniques to assist design efforts relating to a multi-satellite deployment platform. After that, in 2019, I am most grateful to Dr. Boyang Chen of TU Delft for his help in identifying a tractable project that would potentially contribute to some of the theoretical underpinnings of validating the use of neural networks in structural design. That project turned into this thesis.

My thanks go well beyond that initial mentoring. Dr. Chen has been generous with his time throughout, and I thank him for comments on earlier drafts as well as for proofing and helping debug some of the code written in support of this thesis. This thesis also partially builds on earlier thesis work by Tom Gulikers of TU Delft, who considered neural networks applications in user-material subroutines (UMAT). I thank Tom for his time at early stages of my project in helping me understand his approach and conclusions. My own thesis took me to subjects and directions that I had not necessarily anticipated at the outset. For example, I had to learn FORTRAN so that my

Python script could work with legacy FORTRAN routines used in ABAQUS. I am grateful to Giorgio Balducci for reviewing parts of my FORTRAN code. I also acknowledge the kindness of Max Roest of FloatScans in offering to test and print - at no cost - his new 3D printers on a few of my structures, including one complex organic-inspired structure. I also thank all of the staff at TU Delft Faculty of Aerospace Engineering for their office support while I was working out of the faculty before we moved to a home-based working protocol. Finally, I thank my parents and my partner Rhythm for proofing parts of this thesis, and for their unfailing moral support. As always, however, any errors of omission or commission in this thesis are my own.

Abstract

In this thesis, a new data-driven finite element is developed, which is referred to as a neuromorphic element (designated as NmT2). Its goal is to reduce the computational expense of FEA models without compromising solution accuracy by embedding a neural network, trained on an element level. The neural network is developed such that the traditional trial-and-error approach to determining its hyperparameters may be bypassed. This is achieved through a multi-objective optimization algorithm that builds networks with random configurations and uses Latin Hypercube sampling to test them on a fraction of the overall data repository. Once the algorithm reaches a state of diminishing returns over the development of multiple networks, the program is halted and the best performing neural network is saved. The resultant network is then trained over the entire data repository consisting of over half a million datasets. The entire process of a self-designing neural network is called a neuromorphic engine.

The neuromorphic engine is designed to determine the local nodal force vector of a truss member based on the structure's geometry and axial nodal displacements. Axial tension and compression are the two modes of loading that are considered and are pushed to the nonlinear regimes by including post-buckling and material plasticity. In addition, the user is provided with the option of including structural defects in the truss members. Once trained, the neuromorphic engine can be inserted within a user-element subroutine and deployed in ABAQUS.

The neuromorphic element is essentially a truss element which includes the deformation capabilities of beam elements. Unlike traditional FEA methods requiring multiple beam elements, a single NmT2 element can be used when meshing a truss member to model complex behaviour such as post-buckling deformation. To test the capabilities of the neuromorphic element, three case studies are designed as a proof of concept, comparing the performance of NmT2 elements against traditional FEA elements (T2D2 or B22). Overall, the NmT2 elements managed to accelerate the computing time of an FEA model by up to 1,000%, while maintaining solution accuracy within 5%. These results affirm the potential of neural networks within active FEA simulations in the field of data-driven computational mechanics as a means to define complex nonlinear element formulations.

Contents

Declaration of Authorship **iii**

Graduation Committee **v**

Preface **xi**

Abstract **xiii**

List of Figures **xix**

List of Tables **xxiii**

1 Introduction **1**
- 1.1 Aim and scope . . . 2
- 1.2 Thesis layout . . . 3

2 Neural Networks: An Introduction **5**
- 2.1 Defining the neural architecture . . . 6
  - 2.1.1 The perceptron . . . 6
  - 2.1.2 Multilayered neural networks . . . 11
- 2.2 Training a neural network . . . 13
  - 2.2.1 Loss functions . . . 13
  - 2.2.2 Backpropagation . . . 15
- 2.3 Advanced topics . . . 18
  - 2.3.1 Overfitting . . . 18
  - 2.3.2 Hyperparameter optimization . . . 20
  - 2.3.3 Pretraining . . . 20
  - 2.3.4 Morphable neural networks . . . 22
  - 2.3.5 Momentum . . . 23
  - 2.3.6 Additional noteworthy topics . . . 25
- 2.4 Closure . . . 27

3 Neural Network Deployment in FEA **29**
- 3.1 Uncoupled neural networks . . . 30
  - 3.1.1 Top-level damage prediction . . . 30
  - 3.1.2 Manufacturing . . . 32
  - 3.1.3 Post-buckling and crash behaviour . . . 32
- 3.2 Coupled neural networks: active FEA reintegration . . . 33
  - 3.2.1 Constitutive modelling . . . 35
  - 3.2.2 Framework development & data management . . . 39
- 3.3 Closure . . . 42

4 Computational Architecture Design **43**
4.1 The appropriate context: biomimicry . . . 43
4.2 Establishing the parameter design space . . . 46
4.2.1 T2D2: The fundamental FEA truss element . . . 46
4.2.2 Standard user-element (UEL) formulation . . . 49
4.2.3 Neural Network user-element (UEL) formulation . . . 51
4.2.4 Geometrical boundaries . . . 54
4.3 Building the global framework . . . 56
4.3.1 System settings . . . 56
4.3.2 Global computational framework . . . 57
4.4 Closure . . . 60

5 Data Generation & Automation **61**
5.1 Boundary conditions . . . 61
5.1.1 Structural boundary conditions . . . 62
5.1.2 Capping the global data density . . . 62
5.2 Axial tension: an overview . . . 64
5.2.1 Engineering vs True stress/strain . . . 64
5.2.2 Building a semi-analytical approximation . . . 66
5.3 FEA Tension model . . . 67
5.3.1 Plasticity in ABAQUS . . . 67
5.3.2 Mesh convergence and refinement . . . 68
5.3.3 Data generation rate . . . 69
5.3.4 Final ABAQUS parameters . . . 71
5.4 Axial compression: an overview . . . 73
5.5 FEA Compression model . . . 75
5.5.1 Linear eigenvalue analysis . . . 75
5.5.2 Initiating the nonlinear buckling analysis . . . 76
5.5.3 Choosing the initial imperfection . . . 78
5.5.4 Overview: final ABAQUS parameters . . . 81
5.6 Automating the data-generation process . . . 82
5.6.1 Software architecture . . . 82
5.6.2 ABAQUS-Python interface: selected notes . . . 83
5.7 Closure . . . 86

6 Building the Neuromorphic Engine **87**
6.1 Data formatting & preparation . . . 87
6.1.1 Data normalization . . . 88
6.1.2 Data segregation . . . 89
6.2 Morphability: A self-designed neural architecture . . . 89
6.2.1 R-squared Metric . . . 90
6.2.2 Hyperparameter selection . . . 91
6.2.3 Chosen architecture . . . 92
6.3 Training the neuromorphic engine . . . 93
6.3.1 General performance . . . 93
6.3.2 Compounded error analysis . . . 95
6.4 Closure . . . 100

7 NmT2: A New Class of Element 101
7.1 Python - FORTRAN translator . . . 101
7.2 NmT2 Deployment . . . 103

8 Case Study 1: Single Truss Member 105
8.1 Preparing the analysis . . . 106
8.2 Results . . . 107
8.2.1 Tension . . . 107
8.2.2 Compression . . . 109
8.2.3 Global analysis . . . 110
8.3 Closure . . . 114

9 Case Study 2: V-Shaped Structure 115
9.1 Results . . . 116
9.1.1 Scenario 1: Downward loading . . . 116
9.1.2 Scenario 2: Upward loading . . . 121
9.1.3 Scenario 3: Horizontal loading . . . 124
9.2 Closure . . . 128

10 Case Study 3: Howrah Structure 129
10.1 Results . . . 131
10.1.1 Scenario 1: No defect seeding . . . 131
10.1.2 Scenario 2: Including defect seeding . . . 134
10.2 Closure . . . 137

11 Discussion 139
11.1 The neuromorphic engine: Bypassing the trial-and-error approach . . . 139
11.2 Assessing the practical application of the NmT2 element . . . 140
11.3 Final summary: Addressing the initial research questions . . . 144

12 Future Research 147
12.1 Neural network development . . . 147
12.2 Neuromorphic element development . . . 148

13 Closing Remarks 149

Bibliography 151

List of Figures

2.1 Biological neuron [127]. 6
2.2 Artificial neuron [127]. 6
2.3 Pre-activation and post-activation values within a neuron (recreated from [5]). 7
2.4 Various activation functions. 10
2.5 Example of a neural network with two hidden layers and multiple outputs [6]. 11
2.6 Feed-forward neural network with two hidden layers and one output layer, with and without bias neurons (recreated from [5]). 12
2.7 Feed-forward neural network using scalar and vector architectures (recreated from [5]). 12
2.8 Decoupled vector-centric representation of backpropagation (recreated from [5]). . . . 17
2.9 Example of overfitting [16]. 18
2.10 Multilayer autoencoder used for pretraining using dual-level reduction [5]. 21
2.11 Synaptic weights between one layer containing 2 neurons, and an output layer. 22
2.12 Example of a boosting scheme using AdaBoost [7]. 27

3.1 Proposed workflow to train uncoupled finite element-based neural networks (recreated from [65]). 30
3.2 Modified workflow for active FEA reintegration. 34
3.3 Neural architecture used for developing a constitutive model of biaxially loaded concrete [38]. 35
3.4 Autoprogressive training overview [39]. 36
3.5 The average and standard deviation of critical parameters 37
3.6 Neural network training to develop a constitutive model from finite element training data [39]. 38
3.7 Multiscale finite element approach using neural networks embedded at the integration point-level [45]. 39
3.8 Graphical overview of the self-clustering analysis [82]. 40
3.9 Overview of global framework developed by Bessa *et al.* [15]. 41

4.1 Example of a topology optimization process conducted on a bracket. The resultant structure (far right) is capable of withstanding the original loads and boundary conditions, but is lighter than the initial design (far left) [58]. 44
4.2 Organic structural lattice of a leaf [29]. 44
4.3 Porous bone structure [111]. 44
4.4 Example of the practical usage of a neural element. Far left: initial planar multi-truss structure. Middle: traditional FEA discretization (traditional FEA shown in blue). Right: discretization using neural elements (neural element shown in green). 45
4.5 Truss element in 2D space, in local ($x-y$) and global ($X-Y$) reference frames (recreated from [107]). 47
4.6 Example of a truss element subjected to combined loading causing an elongation and rotation. 47
4.7 Flowchart of the solver process within ABAQUS using an interfaced user-element (UEL) subroutine. 50

4.8 Example of a possible neural network within a UEL to compute the internal force vector as a function of the element geometry and nodal displacements. 51
4.9 Flowchart of the solver process within ABAQUS using the interfaced neural element. . . 53
4.10 Example of a metal 3D-printing process using powder bed fusion [2]. 54
4.11 Possible product resulting from a metal 3D-printed process (rendered in KeyShot). . . . 54
4.12 Overview of a truss element's geometrical variables (imposed constraints are greyed-out and cannot be altered by the end-user). 55
4.13 Global computational framework to build, deploy and test the neural element on practical case studies. 59

5.1 Structural boundary conditions imposed throughout the data generation process. . . . 62
5.2 Effect of an axial displacement on a truss structure where $\Delta a_{e,\text{axial}} > 0$. 64
5.3 Example of a stress-strain curve for a isotropic material subjected to tensile loading. . . 65
5.4 Example of mesh refinement for a plate with an open hole using varying sizes of triangular and quadrilateral elements. Note that the mesh density increases closer to the hole to capture the highly-varying stress field [30]. 68
5.5 Minimal mesh discretization of a beam into 3 elements to include the presence of a structural defect, represented by $r_{e,D}$. 69
5.6 The average and standard deviation of critical parameters 69
5.7 Maximum load increment step refinement of f_{e,int_1} vs. $\Delta a_{e,\text{axial}}$ for a randomly generated truss member where: $[L_e, r_e, r_{e,D}]$ = [30.00mm, 1.50mm, 1.20mm]. 70
5.8 The average and standard deviation of critical parameters 72
5.9 Simply supported beam under axial compressive load P. As the axial load is increased, the structure will eventually reach a point where it buckles out of the axial plane (shown in orange). 73
5.10 Effect of mesh discretization and element selection on the truss member's numerically calculated critical buckling load (P_{cr}) using B21, B22, and B23 beam elements. 76
5.11 Overview of the interaction between linear and nonlinear models to conduct a nonlinear buckling analysis. 77
5.12 Effect of imperfection scaling factors ζ on initiating a post-buckling analysis for a simply supported beam where: $[L_e, r_e, r_{e,D}]$ = [30.00mm, 2.00mm, 2.00mm]. Material plasticity is neglected. 79
5.13 Effect of imperfection scaling factors ζ on initiating a post-buckling analysis for a simply supported beam where: $[L_e, r_e, r_{e,D}]$ = [30.00mm, 2.00mm, 2.00mm]. Material plasticity is included. 80
5.14 Code architecture designed to automate the data-generation process. 85
5.15 Example of the full deformation domain including material plasticity and post-buckling within the prescribed boundaries for specimen of length and nominal radius: L_e = 30.00mm, r_e = 1.80mm. All the defective configurations within the parameter design space are considered. 86

6.1 R-squared (R^2) evolution over the training process for training and validation datasets. . 94
6.2 Mean-squared-error ($\mathscr{L}_{\text{MSE}}$) evolution over the training process for training and validation datasets. 94
6.3 Post-training error map between the neuromorphic engine and FEA models across all 4,032 models after 10 epochs. 95
6.4 Post-training error map between the neuromorphic engine and FEA models across all 4,032 models after 500 epochs. 96
6.5 Post-training averages errors for selected data groups. 97
6.6 The average and standard deviation of critical parameters 99

7.1 Interaction between the model input file and neuromorphic element NmT2 prior to being submitted to the ABAQUS solver. 103

8.1 Case study 1: simply-supported beam with displacement loading (in this case, the structure is loaded in tension). 105
8.2 Case study 1: simply-supported beam with displacement loading (in this case, the structure is loaded in compression, note the use of extra nodes to model accurately the post-buckling deformation). 105
8.3 Case study 1: simply-supported beam with displacement loading using a single NmT2 element that has the same geometry as the truss member. 106
8.4 Flow process to prepare the FEA analyses. One using traditional elements, and the other using a single NmT2 element. 106
8.5 Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (either T2D2 or B22 shown in blue), and the same models meshed with a single NmT2 element (shown in orange). . 113

9.1 Case study 2: V-shaped structure of two truss members and loaded at its lower-most node. 115
9.2 V-shaped structure consisting of 2 truss members joined via a pin-connection with downward vertical loading to induce tensile deformation in both members, meshed with T2D2 elements. Heat map represents displacement magnitude. 117
9.3 Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (T2D2), and the same models meshed with two NmT2 elements (shown in orange) for loading scenario 1 applied to a V-shaped structure (2 truss members). 120
9.4 V-shaped structure consisting of 2 truss members joined via a pin-connection with upward loading to induce buckling in both members, meshed with B22 elements. Heat map represents displacement magnitude. 121
9.5 Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (B22), and the same models meshed with two NmT2 elements (shown in orange) for loading scenario 2 applied to a V-shaped structure (2 truss members). 123
9.6 V-shaped structure consisting of 2 truss members joined via a pin-connection with horizontal loading to induce differential loading in both members, meshed with B22 elements. Heat map represents displacement magnitude. 124
9.7 Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (B22), and the same models meshed with two NmT2 elements (shown in orange) for loading scenario 3 applied to a V-shaped structure (2 truss members). 126
9.8 Displacement profiles between B22 and NmT2 elements of the structures extracted from Figure 9.7 of the bottom-most node in the V-structure showing the correlation between the horizontal and vertical displacement (U_x and U_y) due to the applied horizontal load. 127

10.1 The Howrah bridge in Kolkata, spanning over the Hooghly river [97]. 129
10.2 Overview of the Howrah structure consisting of 13 truss members. The structure is pinned at node 1 and simply supported at node 5. A downwards displacement load (ΔU_y) is applied at node 3. Throughout this case study, load-displacement curves are built by plotting the reaction force at node 1, f_{e,int_1}, versus ΔU_y. 130
10.3 Example of a meshed Howrah structure in ABAQUS using B22 elements. Node labels were left out for clarity. 131
10.4 Example of a rendered Howrah structured in ABAQUS with no defect seeding. 132

10.5 Model 1. Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. No defect seeding. 133
10.6 Model 2. Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. No defect seeding. 133
10.7 Model 3. Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. No defect seeding. 133
10.8 Example of a rendered Howrah structured in ABAQUS with defect seeding. 134
10.9 Example of a loaded Howrah structured in ABAQUS with defect seeding, which buckles in truss member 5 due to structural defects. 134
10.10 Model 1. Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Random defect seeding included. 136
10.11 Model 2. Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Random defect seeding included. 136
10.12 Model 3. Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Random defect seeding included. 136

11.1 Case study 1: simply-supported beam with displacement loading (in this case, the structure is loaded in tension). 141
11.2 Case study 2: V-shaped structure of two truss members and loaded at its lower-most node. 142
11.3 Case study 3: Howrah structure consisting of 13 truss members. The structure is pinned at node 1 and simply supported at node 5. A downwards displacement load (ΔU_y) is applied at node 3. 142

List of Tables

2.1 Backpropagation schemes corresponding to certain activation functions between the ith and $(i+1)$th layer [5]. 17

4.1 Neural network inputs and their respective boundaries (imposed constraints are greyed-out and cannot be altered by the end-user). 55
4.2 Software used throughout the thesis. 56
4.3 Hardware used throughout the project. 57

5.1 Neural network input variables and their respective boundaries including iteration boundaries. 63
5.2 Data conversion process from nominal to true values from experimental strain values corresponding to steel under axial tension with $E = 210$GPa and $\nu = 0.28$. The green columns represent the final inputs into ABAQUS' plastic material behaviour [4]. 67
5.3 Essential ABAQUS settings for the tension model generator. 71
5.4 Essential ABAQUS settings for the compression model generator. 81

6.1 Data normalisation boundaries for all parameters flowing into and out of the neural network. Recall that L_e, r_e, $r_{e,D}$ and $\Delta a_{e,\text{axial}}$ are inputs to the network and $f_{e,\text{int},1}$ is the projected output. 88
6.2 Chosen hyperparameters to optimise. 91
6.3 Fixed hyperparameters throughout the optimisation algorithm. 92
6.4 Performance metrics from the developed algorithm which selects an optimal configuration of hyperparameters on a randomly sampled dataset from the global repository. . . 93
6.5 Selected hyperparameters. 93
6.6 Key final performance metrics of the trained neuromorphic engine after 500 epochs. . . 94

8.1 Error and temporal analysis for all tensile cases between traditionally meshed models using T2D2 elements and models using a single NmT2 element. 108
8.2 Error and temporal analysis for all compressive cases between traditionally meshed models using B22 elements and models using a single NmT2 element. 109
8.3 Error analysis for all cases between traditionally meshed models using standard FEA elements (either B22 or T2D2) and models using a single NmT2 element. 112
8.4 Computational efficiency of traditional FEA elements versus a singular NmT2 element for case study 1. 114

9.1 Error and temporal analysis for loading scenario 1 between traditionally meshed models using T2D2 elements and models using two NmT2 elements. 119
9.2 Error and temporal analysis for loading scenario 2 between traditionally meshed models using B22 elements and models using two NmT2 elements. 122
9.3 Error and temporal analysis for loading scenario 3 between traditionally meshed models using B22 elements and models using two NmT2 elements. 125
9.4 Computational efficiency measured in terms of computing/processing time of traditional FEA elements versus two NmT2 elements for case study 2. 128

10.1 Error and temporal analysis across three models with no defect seeding. 132
10.2 Error and temporal analysis across three models with random defect seeding. 135
10.3 Computational efficiency of traditional FEA elements versus 13 NmT2 elements for case study 3. 137

11.1 Final global hyperparameters of the neuromorphic engine. 140
11.2 Overview of key metrics between all three case studies. Absolute errors and R^2 metrics refer to the amount of deviation between FEA models meshed with NmT2 elements relative to FEA models meshed with traditional elements (T2D2/B22). The gain in efficiency and loss in accuracy relative to FEA models meshed with traditional elements (rows in red) are averaged over all scenarios for a given case study. 143
11.3 Estimated development cost of the NmT2 element assuming no learning curve is required. 146

1 Introduction

Let us consider the chair. The chair is composed of multiple components: minimally a seat, and front and rear legs. Your chair may also have an apron as well as a back made of two stiles, a cross rail, and (perhaps) a few spindles. All these components are assembled together to constitute a piece of furniture which has to support our weight under various conditions. The moment we sit down on the chair, the entire structure becomes loaded under a vertical downward force. The front and rear legs will be loaded under compression to compensate for our body weight. If we lean back, then the back support of the chair and stiles could be subject to bending loads. Regardless of how we position ourselves, the chair's integrity must not fail. If one or more of the structural components were removed or slightly damaged, we might fall to the floor.

The purpose of this analogy is to convey that a structure is composed of multiple components which are interconnected to form a complete product. If the overall structure is loaded in any configuration, then the loads are propagated through the individual structural components. Some components might be loaded in tension, others in compression, some in bending, and perhaps some in torsion. Analyzing the structural components or groups of components that make up a structure, rather than directly analyzing the full structure, is known as *substructuring*. The process of substructuring is fundamental to the work in this thesis. It permits us to divide complex structures into their constituent parts; an analysis of those constituent parts permits us to understand the behaviour of the larger structure.

The design of a sturdy chair requires us to understand how its individual structural components respond and deform when loaded. Computational structural mechanics is a discipline that revolves around the study of structural phenomena governed by mechanical principles using computational methods. In the chair example, such methods could be used by structural engineers to determine the stresses and deformation of all the components and whether any of them are prone to failure. One of the most used tools for substructuring work is *Finite Element Analysis* (FEA) software that uses the *Finite Element Method* (FEM) to compute the mechanical behaviour of structures under various loading schemes. The numerical mathematics used throughout these processes allow engineers to design and analyse structures, with complex boundary conditions, such that they can be modelled as realistically and accurately as possible.

In the context of aerospace engineering, computational structural mechanics is a key aspect present in any design and optimization cycle. Designing sound structures that meet the exigent safety tolerances of the aerospace industry all while minimizing the overall mass is a permanent conundrum for engineers. Often costly, high-fidelity FEA numerical models are required. There is a constant trade-off during an FEA process of accuracy versus computational efficiency: the higher the degree of accuracy the user wishes to achieve, the longer it takes to run the simulation. Ideally, engineers have cluster-based computing systems and are not restricted in terms of hardware, meaning that extensive amounts of computational processing power can be entirely off-loaded for a single FEA analysis. Even with this, numerical models might take hours, days or even weeks to converge to an acceptable solution. An alternative to accelerate the process is to halt the analysis and judge when the results are sufficient, or "good enough". This, however, requires both experience with the software, and competence in the particular engineering field. Such expertise may not be readily available. An alternate idea is to internally change the computational process of the FEA model to accelerate the speed at which it reaches a solution, without requiring excess hardware or computational power. This is when machine learning can be of help.

Machine learning is a field of study focused on the belief that machines are capable of interpreting and analyzing data and, from there, make observations and decisions on their own as well as future predictions, without human assistance. The underlying drive behind this field is data. By continuously providing machines with the appropriate data and the proper environment to learn from the data, they are capable of extraordinary feats. Although machine learning dates back to the 1940s, its popularity exploded in the 1990s with the introduction of the *deep neural network*. The expressive and predictive capabilities of neural networks are highly promising for engineering applications. Given that they are purely data-driven processes, there are no complex computations or equations that are continuously being solved. Introducing machine learning, and neural networks in particular, within computational structural mechanical processes such as FEA, could improve significantly computational efficiency and entirely reduce the need for extravagant computing hardware and resources.

1.1. Aim and scope

The overall aim of this master thesis is to demonstrate that traditional FEA analyses can be accelerated by incorporating machine learning techniques, notably a deep neural network, while maintaining solution accuracy. The overall aim is to improve computational efficiency without compromising solution accuracy. To that end, a new kind of finite element – the neuromorphic element – is developed: it uses a neural network at specific points within the user-subroutine to accelerate expensive computational tasks. The new element is then built into an FEA model such that it may be used during an active simulation. Given the level of expressiveness that can be achieved using a properly trained neural network, the new element attempts to model more complex behaviour (such as material plasticity and post-buckling), which would otherwise require complex mesh discretization schemes and higher order elements. Regarding the development of the neural network itself, choosing the appropriate architecture to model and to extrapolate the dataset in a high-fidelity manner is typically achieved through a trial-and-error approach. Although such an approach might be valid in simpler settings, it is considered extremely inefficient, unreliable, and paradoxically defeats the purpose of building an intelligent system. Therefore, building a neural network that selects its own architecture is of high importance throughout this project. The contents of this thesis ultimately strive to showcase the potential of data-driven computational mechanics versus traditional FEA methods. With this in mind, the following research objective may be formulated:

The objective within the time-span of the thesis is to reduce the computational expense of a finite element analysis – without compromising its accuracy – by embedding neural networks trained on an element level into an active mesh.

A selection of research questions (RQ) are also devised to guide the overall research objective:

RQ1 Is it possible to develop an intelligent framework for neural network design such that the user does not need to manually select and test all parameters to create a high-fidelity model?

RQ2 Is it feasible to embed a trained neural network into an active finite element analysis? Furthermore, can the neural network-based elements work together without disrupting the rest of the FEA environment?

RQ3 Can a neural network-based finite element accurately capture highly nonlinear phenomena such as plasticity and post-buckling that would otherwise require complex FEA models?

RQ4 If the previous question is answered positively, does the change in computational expense justify the development of complex neural network-based systems for FEA modelling?

1.2. Thesis layout

Chapters 2 and 3 contain the literature review for this project. Chapter 2 focuses on neural network development, whereas Chapter 3 investigates neural network usages in the context of structural mechanics and FEA interaction. Chapter 4 establishes an overall computational architecture, and defines the key aspects of the development of a custom finite element that uses an embedded neural network. The extensive data generation strategy is outlined in Chapter 5, and the development of the neural network is presented in Chapter 6. Chapter 7 represents the last of the preparatory phase and explains how the neural network is integrated within a custom finite element and can be deployed in a FEA analysis. Case studies to test the validity of the new element and how it measures up to traditional FEA processes, are presented in Chapters 8, 9 and 10. A final summary and discussion is presented in Chapter 11 to review the entirety of the project, including the four research questions noted above. The discussion of the research questions is informed by the analytical findings of the thesis (for RQ1, RQ2, and RQ3); the author's personal judgement and experience during this period are used to provide an opinion on whether further use of the approach is justified (for RQ4). Suggestions for future research are treated in Chapter 12. The thesis closes in Chapter 13 with the author's personal thoughts on the use of data-driven methods applied to computational structural mechanics, as well as to other applications in general.

2 Neural Networks: An Introduction

Machine learning is a scientific field under the artificial intelligence (AI) umbrella that focuses on developing computer models to perform specific tasks without requiring explicit instructions. Given the current era of Big Data, machine learning can capitalize on the large amounts of information being quantified, collected and stored on a daily basis to help streamline tasks across countless industries. Neural networks are one of the dominant machine learning techniques that are employed for their expressive and predictive capabilities when applied to complex problems. For example, unsupervised neural networks are the backbone of algorithmic high-frequency trading (stock market trading which aims to profit from minute price shifts over the course of milliseconds), which accounts for more than half of equity shares traded on US markets [56]. Self-driving vehicles is another emerging technological field that relies on convolutional neural networks to steer the vehicle in the correct direction while ensuring passenger safety. The capabilities of these smart algorithms will infiltrate every industry within the coming decade to help companies maximize their effectiveness, whether it be improving marketing schemes or manufacturing processes.

Within the context of engineering, machine learning is not as prominent as in other fields. The reasons for this can be traced back to safety and risk tolerances, where the idea of a neural network introduces skepticism and uncertainty. This stems from a gap between the computer science and engineering worlds, which will be discussed at the end of this thesis. Despite the scarce usage of machine learning in engineering, it has become more widespread over the past decade in two specific areas: highly detailed numerical optimization, and high level design. Finite element modelling is one example of the former, and it is the focus of this literature study and thesis.

The finite element method (FEM) is a numerical approach that is widely used to solve highly complex engineering problems that cannot be solved analytically. In aerospace engineering, it is most used in computational solid and fluid mechanics. In either case, engineers must often compromise between the time it takes to run a finite element simulation (which can quickly become of the order of days in the case of complex problems), and the reliability of the model. The scope of this literature review pertains to the solid domain where neural networks are starting to be used in conjunction with finite element software (such as ABAQUS or NASTRAN), to improve computational efficiency without compromising accuracy. This section provides an introduction to neural networks in the structural engineering sector.

2.1. Defining the neural architecture

Numerical neural networks are essentially based on their biological counterpart: the human brain. The cells within the human nervous systems are referred to as *neurons* (or *nuclei*), which are connected to one another via *axons* and *dendrites*. The regions between the latter two are called *synapses* [5]. Both the biological and artificial neurons are illustrated in Figures 2.1 and 2.2. As may be seen, the artificial neuron has a similar architecture to the biological one. Inputs of the form of discrete values $(x_1, x_2, x_3, ..., x_n)$ are projected along singular paths that are each attributed a *synaptic weight* $(w_1, w_2, w_3, ..., w_n)$ culminating at a summation point. The sum of all the inputs are then passed through an activation function before being output. In both instances, neurons are combined to form a multi-nodal network leading to the central nervous system in the case of the human body, or an artificial neural network in the case of computer science.

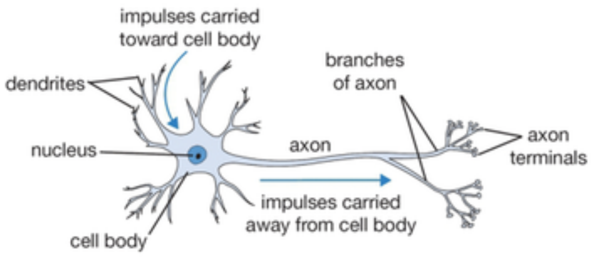

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure 2.1: Biological neuron [127].

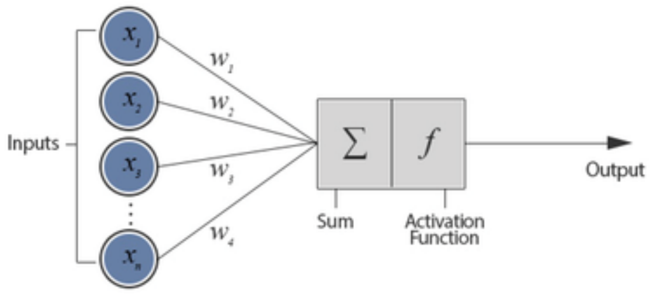


Figure 2.2: Artificial neuron [127].

The workings of a neuron were first proposed in 1943, by neurophysiologist Warren McCulloch and mathematician Walter Pitts [87]. Although their research was focused on modelling a biological neuron in the brain, it was used as a stepping stone in 1958 by Frank Rosenblatt to develop the perceptron algorithm, the basis for all neural networks [101]. Despite these advancements, it can be argued that Alan Turing was the first to radically suggest that an intelligent computer system similar to that of the human brain could be developed. However, this requires that the machine be subjected to an appropriate course of education, thereby mimicking the evolution of a child's brain into that of an adult's [119]. In any case, the concept of training a machine to think and make conclusions without explicit instructions has been around for more than half a century.

The neural architecture is a key factor that influences the predictive capabilities of the network and can be tailored for each application. It can be broken down into two categories: the perceptron and the layering process.

2.1.1. The perceptron

A perceptron is the simplest possible neural network as it contains only one computational layer, shown in Figure 2.2. The number of inputs can be compiled into an input vector $\bar{X}$ which contains n feature variables such that $\overline{X} = [x_1, x_2, x_3, ..., x_n]$. As each input is passed through its respective axon, it is subjected to a synaptic weight where the global vector has the same dimensions as the input vector: $\overline{W} = [w_1, w_2, w_3, ..., w_n]$. It should be stressed that up until this point, the input layer has not performed any computations on its own. A linear function is then applied to sum all the inputs and weights: $\overline{W} \cdot \overline{X} = \sum_{i=1}^{n} w_i x_i$. The final step is passing this summation through an activation function typically denoted as Φ. Therefore, the output of a perceptron can be written as:

$$\hat{y} = \Phi(\overline{W} \cdot \overline{X}) \tag{2.1}$$

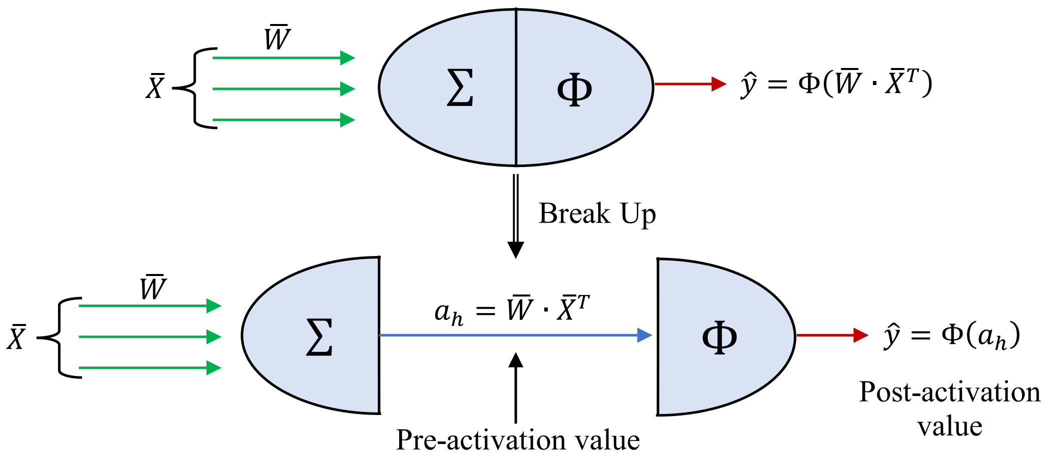


Figure 2.3: Pre-activation and post-activation values within a neuron (recreated from [5]).

An artificial neuron is computing two functions within a node: a summation and an activation, also referred to as pre-activation and post-activation respectively as shown in Figure 2.3. Typically, the focus is always on the output of an artificial neuron, however, the summation stage plays an important role when optimizing the network through backpropagation, which will be discussed in the following section. The final step in defining the architecture of the perceptron is to determine the nature of Φ. The choice of activation function is critical as it ultimately defines the output of a neuron. There exists multiple activation functions and should be carefully chosen based on the application of the network.

Linear activation

$$\Phi(v) = v \tag{2.2}$$

The linear activation (commonly referred to as the *identity function*) is the simplest activation function which provides no nonlinear representation. In a multilayered network, the identity function is sometimes used in the final output layer when the desired target is a real value unconstrained by a scaled domain: $\mathscr{R}(\Phi) \in (-\infty, \infty)$. A simple example would be a perceptron that converts miles to kilometers. The input vector would contain a singular value in miles that is fed into a neuron and then passed through the identity function. Given the linear nature, the neuron outputs a real value on the same scale as the input. The conversion from miles to kilometers is therefore entirely dependant on the synaptic weight. Such a function facilitates simple integration and high computational efficiency, but cannot when nonlinear modelling is required.

Sign

$$\Phi(v) = \text{sign}(v) \tag{2.3}$$

Binary classification problems benefit the most from the sign function (also known as the *binary step* or *heaviside* function) which simply returns a value of either 1 or -1. When confined to the perceptron, this function can be particularly powerful, especially when trained on an extensive dataset. It is currently being extensively implemented within the medical sector to assist doctors in determining whether a tumour is malignant or benign [90]. Unfortunately, the capabilities of this activation quickly break down when embedded in a network due to its non-differentiability.

Sigmoid

$$\Phi(v) = \frac{1}{1+e^{-v}} \tag{2.4}$$

The output of the Sigmoid function will always be between 0 and 1: $\mathscr{R}(\Phi) \in (0,1)$, making it perfectly suited for computing probabilistic quantities. Its nonlinear behaviour and non-zero, continuous derivative have made it one of the most used activation functions in the field of machine learning. However, outputs from a Sigmoid function are not centered around 0, which introduces unfavourable dynamics when optimizing the network through backpropagation [5]. Furthermore, the Sigmoid suffers from a concept known as the *vanishing gradient* meaning that there is practically no variation in the gradient at high and low values of v. These two latter problems will become apparent when discussing the training of a neural network.

Hyperbolic tangent

$$\Phi(v) = \tanh(v) = \frac{e^{2v}-1}{e^{2v}+1} = 2\cdot\text{Sigmoid}(2v)-1 \tag{2.5}$$

The hyperbolic tangent is closely related to the Sigmoid function as shown in Equation 2.5. Graphically, when compared to the Sigmoid, it is horizontally re-scaled and vertically extended such that its outputs are always between -1 and 1: $\mathscr{R}(\Phi) \in (-1,1)$. Given that it is zero-centered, it circumvents the training problems that plague the Sigmoid function, while retaining the nonlinear expressiveness and a continuous non-zero derivative. Therefore, the hyperbolic tangent is usually preferred over the Sigmoid, especially when the outputs of the network are desired to be both positive and negative.

Hard hyperbolic tangent

$$\Phi(v) = \max\{\min[v,1],-1\} \tag{2.6}$$

In recent years, it has been found that piecewise activation functions are more computationally efficient than continuous ones [5]. One example is the hard hyperbolic tangent, which has the same output range as the classical hyperbolic tangent but helps the network converge faster to the desired optimum. An important observation is that $\Phi'(v<1 \vee v>1) = 0$. Inputs corresponding to the aforementioned domain cause the derivative of the function to become zero, which impairs the network from learning through backpropagation. This is referred to as the *dying hard hyperbolic tangent*. Provided that the input data is properly conditioned, it can be easily contained but does require active monitoring. Although this function does help a network converge quickly and is computationally efficient, it is often substituted for its piecewise counterpart: the rectified linear unit.

ReLU (Rectified Linear Unit)

$$\Phi(v) = \max\{v,0\} \tag{2.7}$$

The rectified linear unit is another piecewise activation function that has proven to be highly computationally efficient and is the most used in the design of artificial neural networks. It is very similar to the hard hyperbolic tangent, except that its output is not capped at ± 1. In other words: $\mathscr{R}(\Phi) \in [0,\infty)$. This unique property makes it the sole activation function that shows asymmetric saturation which accelerates the training process. However, similarly to the hard hyperbolic tangent, the ReLU suffers from an undefined gradient for certain input values: $\Phi'(v<0) = 0$. Though in practice, this is usually not problematic as additional neurons within a network compensate for those whose input/output is 0. The ReLU is being increasingly used across a wide variety of applications to the extent that it is almost always present in any given neural network.

Leaky ReLU

$$\Phi(v) = \max\{\kappa \cdot v, v\} \tag{2.8}$$

The leaky ReLU is an attempt to eradicate the dying ReLU problem by providing a small negative slope when $v < 0$. This fixes the zero-gradient problem but does not provide a significant amount of added benefit to the ReLU which is already very computationally efficient.

Swish

$$\Phi(v) = \frac{v}{1+e^{-v}} = v \cdot \text{Sigmoid}(v) \tag{2.9}$$

Swish is another attempt to further improve the ReLU by Google researchers Prajit Ramachandran, Barret Zoph, and Quoc Le. It is closely related to the Sigmoid function and emulates the piecewise nature of the ReLU function, therefore it is simple to manually code into a neural network if the chosen programming environment does not already offer it as a module. Furthermore, its continuous nature eradicates the dying ReLU problem. Performance improvements showed that Swish is on average 0.6% – 0.9% more accurate than ReLU for classification problems without decreasing its convergence speed [99]. Although this might seem like a minute improvement, it does make a difference for high-speed classification tasks that are used in self-driving algorithms.

Softmax

$$\Phi(v)_i = \frac{e^{v_i}}{\sum_{n=1}^{N} e^{v_n}} \qquad \text{for } i = 1, 2, \ldots, N \tag{2.10}$$

Also known as the *normalized exponential function*, Softmax normalizes an N-dimensional input vector into a probabilistic distribution with N probabilities whose sum is equal to 1. This makes it perfectly apt for an output node within a neural network and well-suited for multilayered classification problems.

These activation functions are the most common and are always considered when designing a neural network. Figure 2.4 shows a graphical representation of all the listed activation functions including Softmax, which was created using a randomly generated vector. The ReLU is regarded as the most successful and widely used function due to its computational efficiency [99]. Although some alternatives have been proposed such as the leaky ReLU and Swish, neither of them have managed to entirely replace the ReLU. This, however, does not mean that continuous functions such as the Sigmoid and hyperbolic tangent are no longer in use. They continue to be very useful in conjunction with piecewise functions that work together in deep neural networks.

When choosing the appropriate activation function, not only should the output range and application be considered, but also the derivative of the function. This will become of significant importance during the training process, which relies on backpropagation to optimize a network. This section focused solely on the architecture of the network, but the training stage is an equally important stage that will be elaborated subsequently.

The perceptron is the simplest form of a neural network as it only contains one neuron with one activation function. From here, it is a small step to a neural network which is simply a combination of multiple perceptrons.

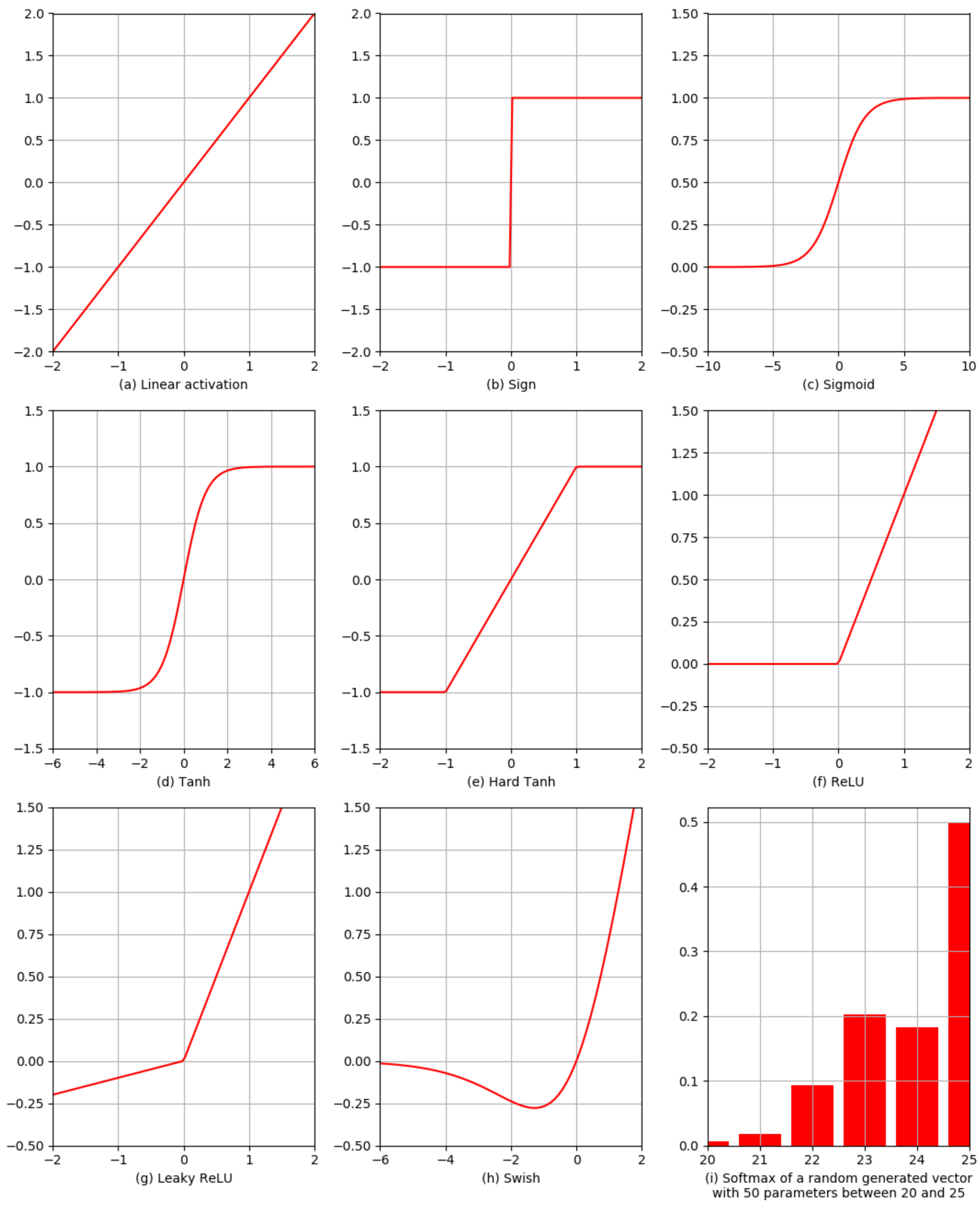


Figure 2.4: Various activation functions.

2.1.2. Multilayered neural networks

Unlike a perceptron, a multilayered neural network (sometimes referred to as *multilayered perceptron* or MLP for short) has multiple computational layers. These contain an input layer, an output layer, and several layers in between called *hidden layers.* Such networks are known as feed-forward networks since each layer feeds into the following one until the output is reached. Therefore, the architecture of a neural network is fully defined when the number of layers and nodes in each layer are set.

Figure 2.5 shows an example of a neural network. Data is first inserted into the input layer which does not perform any computations in its own right. The inputs are directly fed into the first hidden layer, being subjected to a synaptic weight vector $\overline{W_1}$. Each neuron in the hidden layer sums its inputs and then passes them through an activation function denoted as f_1 in the diagram below. It is important to note that although each neuron in a layer could have a different activation function, this is never the case as it would require too much input from the user and would hinder the training process [5]. The outputs of the first hidden layer are fed into the second along with a second synaptic weight vector $\overline{W_2}$. Another activation function, f_2, is used to compute the output of each neuron which is finally passed on to the output layer where a final activation function is used to compute the output of each neuron. It should be noted that in most cases, the activation functions used in the hidden layers are all the same (i.e., $f_1 = f_2$), but are not necessarily the same as those used in the output layer. The reason for this is purely user simplicity. Building a neural architecture is extremely difficult due to the infinite number of variables. Normalizing the activation function across all hidden layers is one boundary constraint that helps streamline the design of the network. In practice, it is also found that networks are more efficient if the same activation function is used in the hidden layers [5].

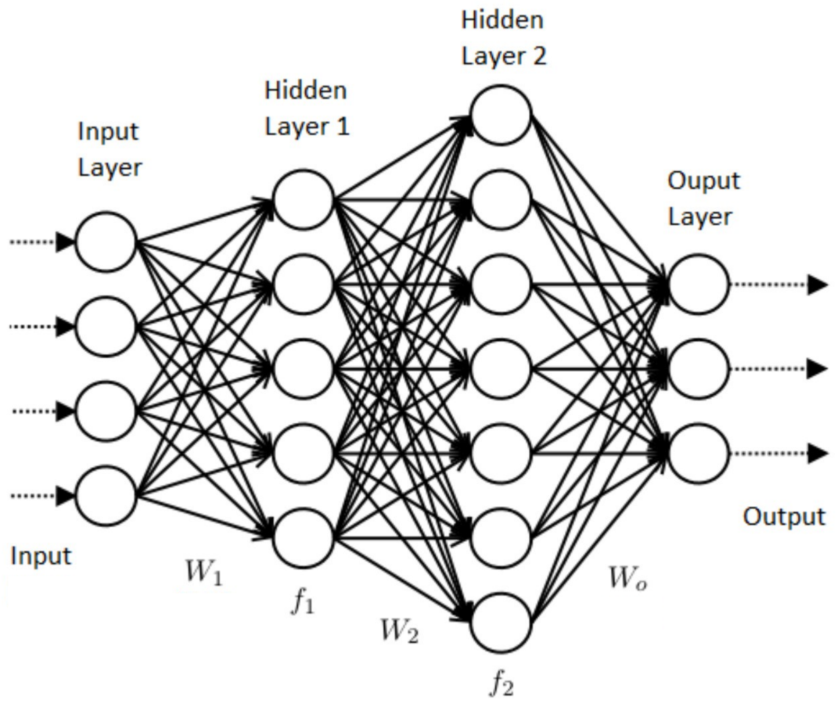
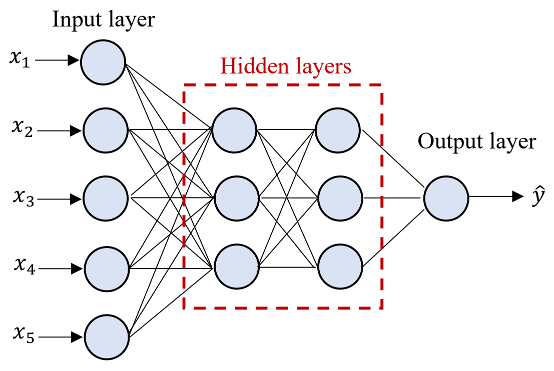
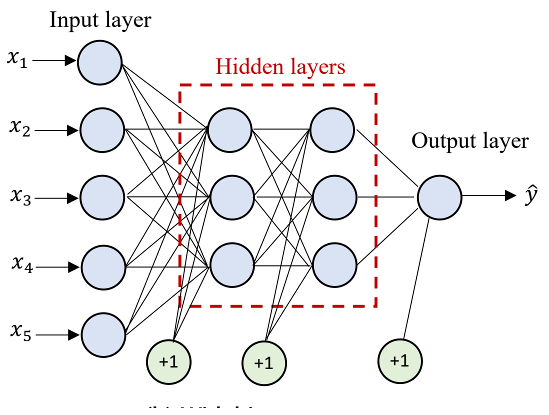


Figure 2.5: Example of a neural network with two hidden layers and multiple outputs [6].

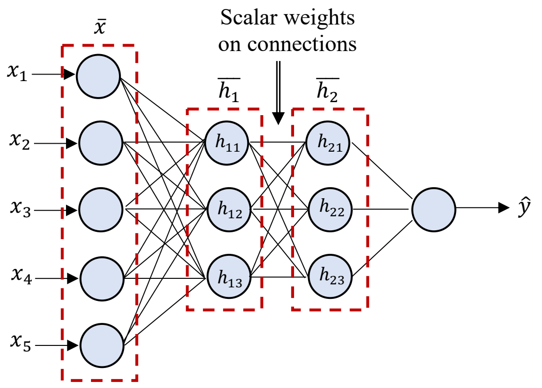


(a) No bias neurons.

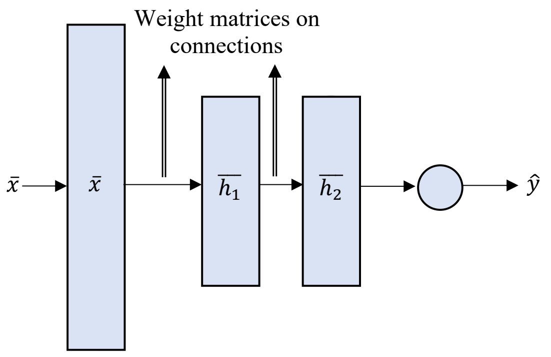


(b) With bias neurons.

Figure 2.6: Feed-forward neural network with two hidden layers and one output layer, with and without bias neurons (recreated from [5]).



(a) Scalar notation.



(b) Vector notation.

Figure 2.7: Feed-forward neural network using scalar and vector architectures (recreated from [5]).

An additional parameter that can be embedded within a neural network is a bias neuron. This is simply an entity that always transmits a value of $b = +1$ to an activation function. Therefore a nodal output with bias would be of the form $\hat{y} = \Phi(\overline{W} \cdot \overline{X} + b)$. Introducing a bias becomes very useful if the desired outputs do not display a trend that passes through the origin i.e. $f(0) = 0$. Figure 2.6 shows the same network, one of which has bias neurons and one that does not. Just like the activation function, a bias neuron could be selectively attributed to specific neurons but in practice, they are globally added to streamline the training process.

One final important concept concerns the notation of a neural network. The number of layers is typically represented by a variable k, and each layer contains $p_1, \ldots, p_k$ neurons, which defines its dimensionality. The column vectors representing the outputs of each layer are denoted as $\overline{h_1}, \ldots, \overline{h_k}$. The weights of the axons between the rth hidden layer and $(r+1)$th hidden layer are contained in a $p_r \times p_{r+1}$ matrix W_r. Vector notation is extensively used when programming a network and is summarized by the following set of recursive equations for a d-dimensional input vector $\overline{x}$:

$$\overline{h}_1 = \Phi(W_1^T \overline{x}) \qquad \text{[Input to Hidden Layer]}$$
$$\overline{h}_{p+1} = \Phi(W_{p+1}^T \overline{h}_p) \quad \forall p \in \{1, \ldots, k-1\} \qquad \text{[Hidden to Hidden Layer]}$$
$$\overline{o} = \Phi(W_{k+1}^T \overline{h}_k) \qquad \text{[Hidden to output Layer]}$$

Figure 2.7 shows a graphical example of a neural network with vector and scalar notation. Scalar notation is straightforward to understand, but is rarely used as vector notation is more computationally efficient.

The number of hidden layers and nodes in each layer is essentially all that is required to define the architecture of a network. The difficulty lies in the reasoning behind these design choices. For instance, adding multiple hidden layers does increase the expressiveness of a network but also encourages overfitting. A phenomenon that paradoxically hinders expressivity. The same logic can be applied to the number of neurons in a layer. Moving from a layer that contains relatively few neurons to one that contains many produces sparsity within the dataset. The opposite compresses the data into a reduced representation. Unfortunately, there is still no formula for designing the optimal neural architecture. Trial and error, and experience in the field of machine learning are what helps the most. However, in practice, a neural network with two hidden layers each containing less than 50 neurons is capable of solving most complex nonlinear problems [5].

2.2. Training a neural network

Up until now, the notions of training and optimizing a neural network have been mentioned several times. This is a pivotal step in designing an effective network as it defines its computational efficiency and effectiveness for a given application.

The dominant method for training a network is known as *supervised learning*. To begin with, a dataset containing inputs and their respective outputs is compiled. The inputs are then fed into the network resulting in a forward cascade of computations across numerous layers, using certain sets of weights. The resulting predicted output of the network, $\hat{y}$ is then compared to the actual output, y, and the error between the two is computed by means of a *loss function*. This error is then propagated back through the network using backpropagation to alter the synaptic weights. This cycle repeats until the error converges below an acceptable tolerance. Once the network is considered sufficiently trained, it can be deployed as a "black box": inputs in the same range as the training set can be input into the network, which generates reliable outputs assured by a convergence criterion during training.

Supervised learning can be split into two areas. The first is determining which loss (or *cost*) function to use to measure the error between outputs, and the second is the backpropagation phase used to update the synaptic weights in the network. This section provides an overview of the essentials of both steps which are the final steps in designing a neural network.

2.2.1. Loss functions

Choosing the correct loss function is a critical step since it defines the output error of a network which directly influences the training process through backpropagation. In the context of a neural network, a loss function should incorporate the individual errors of n training examples. This ensures that a scalar quantity is produced, which can be differentiated with respect to the weight and bias vectors. Furthermore, the loss function should only be determinant on the output layer, thereby leaving the hidden layers untouched. Both conditions are required for backpropagation [5, 93, 96]. The following are some of the most common loss functions in use today. In each case, $y^{(i)}$ and $\hat{y}^{(i)}$ represent respectively the *true* and predicted output of the ith data point.

Hinge loss

$$\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} \max\left\{0, 1 - y^{(i)} \cdot \hat{y}^{(i)}\right\} \tag{2.11}$$

Hinge loss (sometimes referred to as *max-margin objective*) is particularly suited to support vector machines that are used to implement learning methods, as well as training classifiers. The target outputs should be in the domain of $y \in \{-1, +1\}$. One drawback is that the derivative of the hinge loss is undefined at $y^{(i)} \cdot \hat{y}^{(i)} = 1$.

Squared hinge loss

$$\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} \left(\max\left\{0, 1 - y^{(i)} \cdot \hat{y}^{(i)}\right\}\right)^2 \tag{2.12}$$

The squared hinge loss is a simple variation on the classical hinge loss that solves the problem of the aforementioned undefined derivative at $y^{(i)} \cdot \hat{y}^{(i)} = 1$. Other than this, it possesses the same properties and applications as the hinge loss.

Mean squared error (MSE)

$$\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 \tag{2.13}$$

Mean squared error is a widely used function in linear regression. It is versatile in handling multiple types of datasets since it is unconstrained by a set domain. In the context of neural networks, however, it tends to slow down the learning process with asymptotic activation functions, making it only fit for logistic regression.

Mean squared logarithmic error (MSLE)

$$\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} \left(\log(y^{(i)} + 1) - \log(\hat{y}^{(i)} + 1)\right)^2 \tag{2.14}$$

MSLE is closely related to MSE in that it is best suited for performing logistic regression. It is also known for penalizing under-estimates more than over-estimates and can be compared to MSE in the following manner:

1. If $\left(y^{(i)} \wedge \hat{y}^{(i)}\right) \ll 1$, then $MSE \approx MSLE$.
2. If $\left(y^{(i)} \vee \hat{y}^{(i)}\right) \gg 1$, then $MSE > MSLE$.
3. If $\left(y^{(i)} \wedge \hat{y}^{(i)}\right) \gg 1$, then $MSE > MSLE$.

Mean absolute error

$$\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}| \tag{2.15}$$

The mean absolute error can be directly compared with MSE. It too suffers from a slow learning rate with asymptotic functions. Though within a neural network, it tends to perform better than MSE since it does not amplify large errors making it more robust to outliers.

Cross entropy

$$\mathscr{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right] \tag{2.16}$$

Cross entropy measures the divergence between two probability distributions and is commonly used in binary classification problems. As previously mentioned, MSE suffers from a slow training phase when paired with asymptotic activation functions such as Sigmoid. Cross-entropy manages to circumvent this problem making it extremely efficient. Softmax activation with cross-entropy loss is one of the most popular and computationally efficient learning schemes for networks with discrete-valued outputs [5].

Kullback Leibler (KL) divergence

$$\begin{aligned}\mathscr{L} &= \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(y^{(i)}||\hat{y}^{(i)}\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\cdot\log\left(\frac{y^{(i)}}{\hat{y}^{(i)}}\right)\right]\\ &= \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\cdot\log\left(y^{(i)}\right)\right)}_{\textbf{entropy}}\underbrace{-\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\cdot\log\left(\hat{y}^{(i)}\right)\right)}_{\textbf{cross entropy}}\end{aligned} \tag{2.17}$$

The KL divergence was first introduced in 1951 by Solomon Kullback and Richard Leibler as a measure of how a probability distribution diverges to a second expected probability distribution [67]. Essentially, it gauges the predictive power that each sample contributes and has shown to be very effective for training multiple models in parallel by means of ensemble methods such as bagging [131], or boosting [5].

The above-described loss functions are currently the most used in neural network design. When choosing an appropriate loss function, it is vital to consider the application of a network (and thereby its output), as well as the activation function used in the output layer. Failure to take these into account will greatly hinder the training process and reduce computational efficiency. It could also make the network diverge altogether. Once the error at the output of a network is defined, it has to be propagated backward through the network using backpropagation.

2.2.2. Backpropagation

The origins of backpropagation can be traced back to the 1960s where it was used to optimize multistage allocation processes in control theory, especially in the context of flight paths [21, 60]. It was only applied to neural networks in 1974 by Paul Webros [124], although the most recognized form of the backpropagation algorithm is nowadays attributed to Rumelhart *et al.* [102]. The goal of the backpropagation process is to compute the gradient of the loss function with respect to the synaptic weights using differential calculus.

Consider a neural network with k layers, each containing one neuron, h. The designation of any given node is therefore $h_1, h_2, ..., h_k$ and the synaptic weight between two nodes is denoted as $w_{(h_{r-1},h_r)}$. Given a chosen loss function L and that there exists only one path between h_1 and the output node o, then the error with respect to any given synaptic weight is:

$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \frac{\partial L}{\partial o}\cdot\left[\frac{\partial o}{\partial h_k}\prod_{i=r}^{k-1}\frac{\partial h_{i+1}}{\partial h_i}\right]\frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \quad \forall r \in 1, ..., k \tag{2.18}$$

It is important to realize that Equation 2.18 is only valid for a neural network with a singular path from the input to the output node. By incorporating the multivariable chain rule, the gradient along any given path for any given neural architecture can be determined. Amending the previous expression yields (where $\mathscr{P}$ is the set of paths that exist from h_r to o):

$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \underbrace{\frac{\partial L}{\partial o} \cdot \left[\sum_{[h_r,h_{r+1},\ldots,h_k,o]\in\mathscr{P}} \left\{ \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right\} \right]}_{\text{Backpropagation computes } \Delta(h_r,o)=\frac{\partial L}{\partial h_r}} \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \quad \forall r \in 1,\ldots,k \tag{2.19}$$

This expression can be further broken down into two components, the first of which is the path-aggregated term ($\Delta(h_r,o)$). Since a neural network cycles between forward and backward phases, it is possible to determine recursively the aggregation by starting at the output layer and working backward to the input layer [5, 124]. This is sometimes referred to as *dynamic programming* but is often avoided due to the confusion it causes with reinforcement learning (a tool used in unsupervised learning, which is out of the scope of this literature review but is described in-depth by Charu Aggarwal and Coates *et al.* [5, 27]). Considering that a_h is the input value to a hidden unit h just before applying the activation function, $\Delta(h_r,o)$ can be expressed as:

$$\Delta(h_r,o) = \frac{\partial L}{\partial h_r} = \sum_{h:h_r\rightarrow h} \left\{ \frac{\partial L}{\partial h}\frac{\partial h}{\partial h_r} \right\} = \sum_{h:h_r\rightarrow h} \frac{\partial h}{\partial h_r}\Delta(h,o)$$

$$\therefore \quad \Delta(h_r,o) = \sum_{h:h_r\rightarrow h} \Phi'(a_h)\cdot w_{(h_r,h)}\cdot\Delta(h,o) \tag{2.20}$$

The other component in Equation 2.19 is $\frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}}$, which can be computed using the defined activation function:

$$\frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} = h_{r-1}\cdot\Phi'(a_{h_r}) \tag{2.21}$$

Finally, the entire backpropagation phase relies on the initialization of each output node in order to compute the cascading gradients:

$$\Delta(o,o) = \frac{\partial L}{\partial o} \tag{2.22}$$

In the previous expressions, the term a_h was used to break down Equation 2.19 and render it comprehensible for practical applications. Such a term is commonly referred to as the *pre-activation variable*, which is the counterpoint to the post-activation variable h ($h=\Phi(a_h)$). It may sometimes be convenient to express the entire backpropagation process in terms of the pre-activation variables. Therefore, Equation 2.19 can be re-written as:

$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \underbrace{\frac{\partial L}{\partial o}\cdot\Phi'(a_o)\cdot \left[\sum_{[h_r,h_{r+1},\ldots,h_k,o]\in\mathscr{P}} \left\{ \frac{\partial a_o}{\partial a_k} \prod_{i=r}^{k-1} \frac{\partial a_{h_i+1}}{\partial a_{h_i}} \right\} \right]}_{\text{Backpropagation computes } \delta(h_r,o)=\frac{\partial L}{\partial a_{h_r}}} \frac{\partial a_{h_r}}{\partial w_{(h_{r-1},h_r)}} \quad \forall r \in 1,\ldots,k \tag{2.23}$$

Where $\delta(h_r,o)$ can be computed as:

$$\delta(h_r,o) = \sum_{h:h_r\rightarrow h} \left\{ \frac{\partial L}{\partial a_h}\frac{\partial a_h}{\partial a_{h_r}} \right\}$$

$$\therefore \quad \delta(h_r,o) = \Phi'(a_{h_r}) \sum_{h:h_r\rightarrow h} w_{(h_r,h)}\delta(h,o) \tag{2.24}$$

Backpropagation is continuously applied until all the output losses are below an acceptable tolerance. Depending on the choice of neural architecture, the training process will converge anywhere between one and a couple of thousand epochs. From a programming point of view, it is important to understand that backpropagation algorithms are conducted in a decoupled manner. This is shown in Figure 2.8 using vector-based representation.

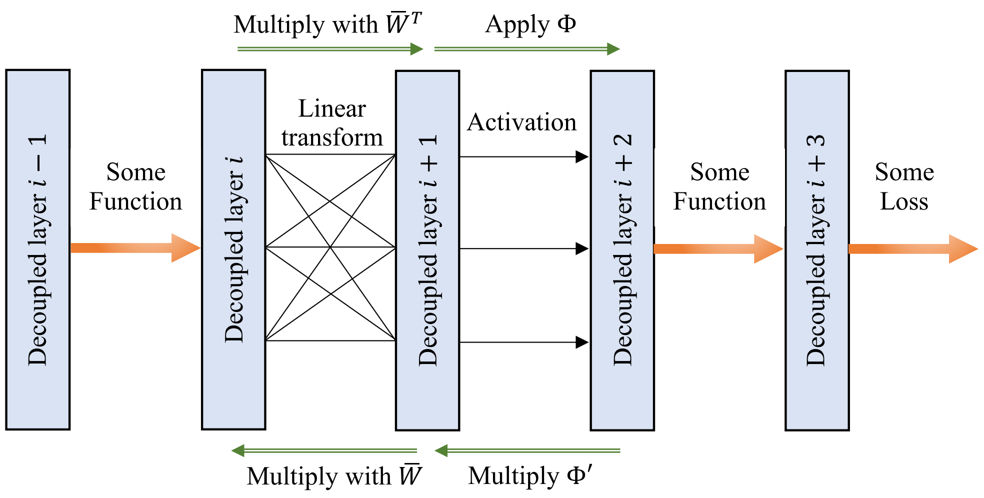


Figure 2.8: Decoupled vector-centric representation of backpropagation (recreated from [5]).

Table 2.1 provides a list of a few functions and their corresponding backpropagation updates between layers. Let $\overline{z}_i$ represent the column vector of outputs of the ith layer, and the synaptic weight vector between layers i and $(i+1)$ is denoted by W. Concerning the backwards phase, $\overline{g}_i$ represents the backpropagated vectors of gradients of layer i. Regarding the notation, it is important to note that $\odot$ is the element-wise multiplication of two vectors i.e. $\mathbf{A}_{n\times 1} \odot \mathbf{B}_{n\times 1} = \mathbf{C}_{n\times 1}$. In the case of an arbitrary function $f_k(\cdot)$, it is required to compute the Jacobian matrix whose elements are:

$$J_{kr} = \frac{\partial f_k(\overline{z}_i)}{\partial \overline{z}_i^{(r)}} \qquad \text{where } z_i^{(r)} \text{ is the } r\text{th element in } \overline{z}_i. \tag{2.25}$$

Numerical optimization by leveraging the magnitude and sign of a derivative is a tool that is widely used in mathematics, computing, and engineering. The effectiveness of a neural network relies on a concrete backpropagation scheme. Unfortunately, numerous problems can arise during training as a result of backpropagation, which will be discussed in the following section. For additional reading, Paul Werbos put forth a book in 1994 detailing the extensive history and uses of backpropagation across all fields in [125].

Table 2.1: Backpropagation schemes corresponding to certain activation functions between the ith and $(i+1)$th layer [5].

Function	**Type**	**Forward**	**Backward**
Linear	Many-Many	$\overline{z}_{i+1} = \overline{W}^T \overline{z}_i$	$\overline{g}_i = \overline{W}\overline{g}_{i+1}$
Sigmoid	One-One	$\overline{z}_{i+1} = \text{Sigmoid}(\overline{z}_i)$	$\overline{g}_i = \overline{g}_{i+1} \odot \overline{z}_{i+1} \odot (1 - \overline{z}_{i+1})$
ReLU	One-One	$\overline{z}_{i+1} = \overline{z}_i \odot I(\overline{z}_i > 0)$	$\overline{g}_i = \overline{g}_{i+1} \odot I(\overline{z}_i > 0)$
Tanh	One-One	$\overline{z}_{i+1} = \tanh(\overline{z}_i)$	$\overline{g}_i = \overline{g}_{i+1} \odot (1 - \overline{z}_{i+1} \odot \overline{z}_{i+1})$
Hard Tanh	One-One	Set to $\pm 1 (\notin [-1,+1])$ Copy $(\in [-1,+1])$	Set to $0 (\notin [-1,+1])$ Copy $(\in [-1,+1])$
Arbitrary function $f_k(\cdot)$	Anything	$\overline{z}_{i+1}^{(k)} = f_k(\overline{z}_i)$	$\overline{g}_i = J^T \overline{g}_{i+1}$

2.3. Advanced topics

The neural architecture and training through backpropagation are the two key elements that are required to build a working neural network. However, they only scratch the surface of neural network design. There are many more concepts that help further improve the training process and a network's efficiency. Even though these topics are more suited to computer science-based applications, this section aims to provide a brief overview of some of the more advanced topics in machine learning. Though it should be noted that most of these are not currently being used for building engineering-oriented neural networks.

2.3.1. Overfitting

Overfitting is one of the most common problems encountered when building a neural network. It occurs when a model can accurately predict the training data but performs poorly when attempting to predict unseen test data, even if the latter lies within the training domain. This unusual situation often occurs when users train their neural networks until the loss error approaches zero ($\mathscr{L} \approx 0$), and then the network displays significant errors after deploying it on testing data. The ability for a network to perform well on unseen test data is known as *generalization*. Possible causes of overfitting are [5]:

- Not using a sufficiently large training set relative to the application.
- Using a complex model (deep neural network) to model a simple problem.
- Using a simple model (shallow neural network) to model a complex problem.
- Training until the error ≈ 0.

With these in mind, designing the architecture of a neural network becomes a delicate balancing act. A network that is too simple will not be able to model the structure of the data, especially in nonlinear datasets; whereas one which is too complex might wrongly represent outliers that should be ignored [18]. From a statistical point of view, an effective network is one that uses an optimal trade-off between high bias and variance (one that is too supple). A focus on high bias is characteristic of an inflexible model; focusing on variance as a determinant in the loss error function typically results in a model which is too supple. A good rule of thumb that is commonly used is that the number of data points used for training should be 2 to 3 times larger than the number of parameters in

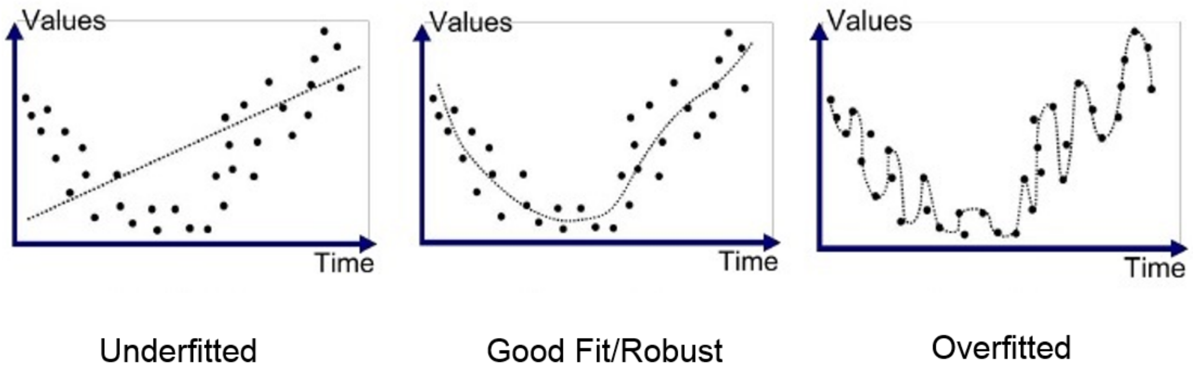


Figure 2.9: Example of overfitting [16].

the neural network [5]. Figure 2.9 shows an example of an overfitted, and underfitted, and a properly calibrated network.

In order to train a network to generalize, it has to be trained to forget: introducing ignorance into a system allows it to ignore noisy patterns and outliers in the data. One way of achieving this is through automated error-based optimization commonly referred to as *regularization*. Here, the variance of a model is controlled by modifying the loss function with a penalty term Ω. Given that the error $E(\overline{X})$ represents the error between the actual and predicted values $(y - \hat{y})$, the updated weight vector including a penalty term is of a given training instance $\overline{X}$:

$$\overline{W} \leftarrow \overline{W}(1 - \alpha\lambda) + \alpha \sum_{\overline{X} \in S} E(\overline{X})\overline{X} \tag{2.26}$$

Where α is the update rate such that $\alpha > 0$, and S is a given batch size. Penalty-based regularization is the most used technique for preventing overfitting and can interestingly be interpreted as adding noise to the training process. Christopher Bishop proves that using penalty terms to regularize a network is equivalent to deliberately injecting noise into the dataset [18]. It has been demonstrated experimentally in 1991 that network generalization can be improved by adding noise [109], Bishop managed to draw parallels with Tikonov regularization (also known as *L2 regularization*) to further streamline the penalty-based process. Although the benefits of training with noise have not yet been demonstrated in engineering applications, they have shown to be extremely effective for improving the accuracy of classifiers.

Another simple solution to prevent overfitting is known as *early stopping*, which simply terminates the training process before it converges to the optimal solution. This is achieved by simultaneously monitoring the error on the training and testing sets. Unfortunately, such a method can only be used if the user already has access to the testing data at the moment of training which isn't the case for most applications. One popular workaround is to randomly split the available data into the training, validation, and testing data whose ratios are usually 50%, 25%, and 25% respectively which have been used since the 1990s [5]. The validation set is used to calibrate overfitting using early stopping to tune other network components such as the neural architecture or backpropagation scheme. Once tuned, the model can be deployed on the test set which has been left untouched. This approach has been shown to improve the generalization capabilities of a network regardless of its application.

Penalty-based regularization and early stopping are two very popular methods for preventing overfitting, yet they induce additional difficulties: in both cases, the user either has more parameters to tune (such as the penalty term and update rate) or is required to carefully monitor the bias-variance trade-off. An ideal solution would help solve the problem of overfitting without burdening the user with more tasks to perform. Such solutions do exist, known as *Dropout* or *pretraining*, but given their complexity, they have been attributed their separate sections. The bias-variance trade-off is a heavily researched area in statistics as well as neural networks. Trevor Hastie, Robert Tibshirani, and Jerome Friedman have compiled an extensive study of this area in the context of machine learning [49]. From a purely algorithmic point of view, Kong *et al.* and Kohavi *et al.* arguably introduced the notion of error correcting through balancing the bias and variance of a system [63, 64], even though their work was not centered on neural networks.

2.3.2. Hyperparameter optimization

Every parameter that the user is required to define to build a neural network such as the number of hidden layers, the number of nodes, the choice of activation functions, etc., is known as a *hyperparameter*. Choosing a good set of hyperparameters can greatly help with a network's generalization capability. Unfortunately, there are no predefined guidelines for determining effective hyperparameters for a given application. In almost all cases, trial and error is used to find the type of network that works best since the number of permutations of a neural network is infinite. *Structural based stabilization* is a process that aims to prevent overfitting by starting off with an effective set of hyperparameters. This is also considered an outer-loop optimization process since it is not yet embedded within an active neural network. The mathematical representation of this process can be expressed as shown in Equation 2.27.

$$\Theta^{(*)} \equiv \underset{\Theta \in \Lambda}{\operatorname{argmin}} \left\{ \Psi(\Theta) \right\} \tag{2.27}$$

Where $\Theta^{(*)}$ represents the desired hyperparameter and Ψ is the hyperparameter response function active across a search space Λ. The response function can be considered as a fictitious parameter since it changes for every application and neural architecture. The most common approach to this problem is to use grid search, which classifies every conceivable combination of hyperparameters within Λ. Several attempts at formulating tailored versions of grid search to improve hyperparameter optimization may be found in [13, 14, 31, 52, 71, 76], though unfortunately these remain extremely theoretical and have yet to be proven within a practical network. Furthermore, it has been shown by Bellman in 1961 that grid search suffers from the *curse of dimensionality*: the number of permutations grows exponentially with the number of hyperparameters [10]. Bergstra *et al.* clearly stated that grid search is a poor choice for building machine learning algorithms and that random set-sampling of hyperparameters is generally more effective and can be easily automated [12].

Hyperparameter optimization remains a very difficult problem, especially when considering neural networks. So far, most solutions that have been proposed are not applicable when building networks that model complex nonlinear phenomena. However, this outer-loop optimization problem is starting to be implemented in simpler applications such as binary-output networks. Thornton *et al.* showed in 2013 that Bayesian optimization could be leveraged to accelerate classification tasks [118]. Their resulting program, Auto-WEKA, proved to be between 35% and 75% faster at classification tasks than conventional algorithms that use grid search or manual search to determine their hyperparameters.

Unfortunately, there has not been any concrete implementation of hyperparameter optimization in complex neural networks, especially for engineering applications. The extensive choice of variables that the user is required to choose is one of the biggest drawbacks of supervised learning. Unsupervised learning, on the other hand, benefits from feature-based learning to automatically tune its hyperparameters. Trial and error using grid search remains the only way to design a neural network for engineering-oriented problems.

2.3.3. Pretraining

Pretraining is another form of regularization that uses an out-of-loop greedy algorithm to find a good initialization point for the synaptic weights prior to training the network. This eliminates one hyperparameter as the user no longer has to decide how to initialize the weights before training. *Unsupervised pretraining* has shown to be the most effective form of pretraining, which has been

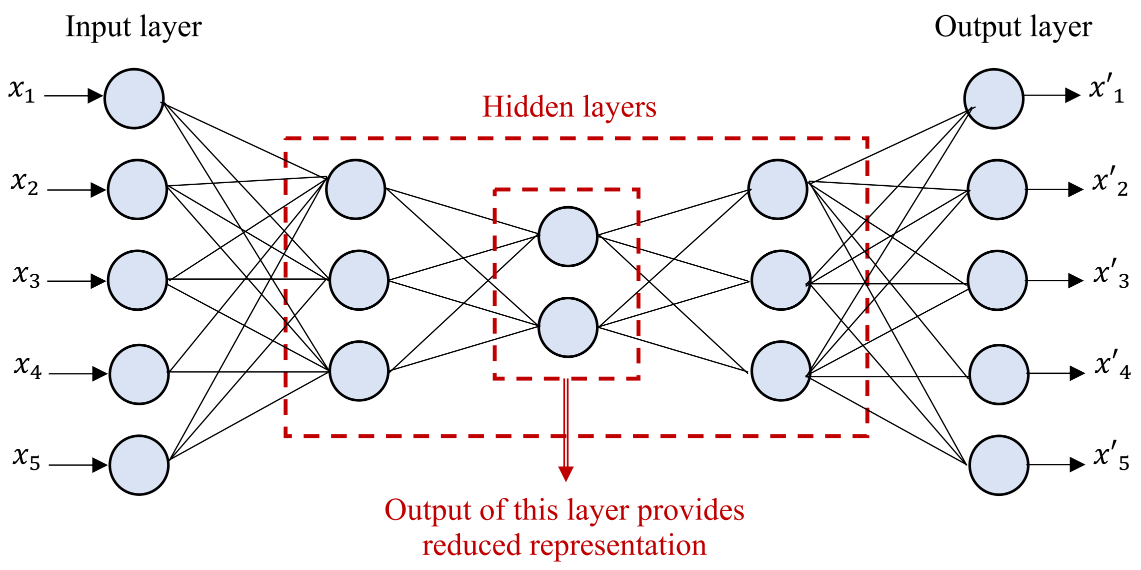


Figure 2.10: Multilayer autoencoder used for pretraining using dual-level reduction [5].

extensively demonstrated in [11, 34]. Supervised pretraining can also work, but typically leverages the greedy algorithm too much and as a result, the weights in the first few layers are already directly related to the output. Unsupervised pretraining is slightly more docile and is able to initialize the synaptic weights such as to minimize the training error and reduce the probability of overfitting.

Unsupervised pretraining relies on dimensionality reduction. Within the neural architecture, this is achieved by reducing the number of nodes from the ith to the $(i+1)$th layer, and then symmetrically increasing the number of nodes in the $(i+2)$th layer as shown in Figure 2.10. This form of neural architecture is also known as an *autoencoder*. Moving data between two layers, the second of which has fewer nodes than the first, consequently reduces the dimensionality of the data. Due to the symmetrical nature of the network, it can be assumed that the outer hidden layers contain a first-level reduced representation of a larger dimensionality, and the inner ones represent a second-level reduced representation but of smaller dimensionality [5]. Such a reduction permits a greedy algorithm to quickly learn the synaptic weights of the network one layer at a time. From there, the weights are used as initialization parameters to fine-tune the neural network through backpropagation. In practice, pretraining can be broken down into the following steps:

1. Consider a random neural architecture with k hidden layers, where each neuron in a hidden layer has an activation function Φ_1. The nodes in the output layer all have an activation function Φ_2. With this in mind, pretraining starts at the end of the network and works towards the beginning layer by layer.

2. The output layer is removed and the representation of the kth hidden layer is learned by creating an autoencoder with $(2 \cdot k - 1)$ hidden layers where the middle layer is the final hidden layer.

3. The resulting architecture of the created autoencoder has $(2 \cdot k - 1)$ hidden layers where the first $(k-1)$ hidden layers are equal to the first $(k-1)$ hidden layers of the original neural network. However in the autoencoder, an additional (k-1) layers are added which are the symmetric counterparts of those already present. The activation functions used in the autoencoders do not have to be related to Φ_1 or Φ_2 (typically linear activations are used since the purpose of an autoencoder is to recreate the input data i.e. $x_1 = x'_1, x_2 = x'_2, ...$).

4. The autoencoder is trained using data from the global training set (the same training dataset that would otherwise be used to train a neural network in the traditional sense).

5. Once trained, the weights from the first k hidden layers can be used to initialize the weights in the original neural network. The weights between the final hidden layer and output layer can also be initialized by training them separately as a cohesive single layered network.

The results of unsupervised pretraining using an autoencoder to initialize the weights of a neural network are quite notable and have shown to be extremely effective in modelling complex problems. Hinton *et al.* showed that significant performance improvements can be achieved as well reducing the training and testing error to well below 1.5% across a wide variety of applications ranging from image recognition to classification [53, 54]. Geoffrey Hinton even goes so far as to state that reducing a dataset's dimensionality is key in designing highly expressive neural networks [54]. An in-depth study on pretraining can be found in [5, 34].

2.3.4. Morphable neural networks

A neural network that automatically alters its own architecture in order to optimize itself is highly sought-after. This would unequivocally eradicate the concept of hyperparameter optimization, as the network would choose the best combination of parameters by itself. Unfortunately, such technology is not yet present but is thought to be an area that large corporations such as Microsoft, Apple, IBM, and Google are investing in heavily. However, there are concepts that have been applied to neural networks to alter and improve their architecture. The most popular one is known as *Dropout*.

Dropout is based on a theoretical dilemma called *feature coadaptation* [108]. Consider the diagram shown in Figure 2.11. Two nodes in a hidden layer transmit their outputs to a singular output node using synaptic weights w_1 and w_2 such that $w_1 \neq w_2$. The error from the output node is then determined and backpropagated to update the weights in the network. Given that the weights are not the same, they will be updated differently. However, if w_1 has already reached its optimal value, then it would be unnecessarily absorbing some of the error that should be entirely diverted to correct w_2. This is known as feature coadaptation. Although it is evidently impossible to know when a certain synaptic weight has reached its optimal value, it introduces the notion that certain weights need not be updated as much as others during training.

The first appearance of enabling a network to self-adapt its architecture in order to prevent feature coadaptation was introduced in 1988 by Sietsma *et al.* and was called *pruning* [108]. If during

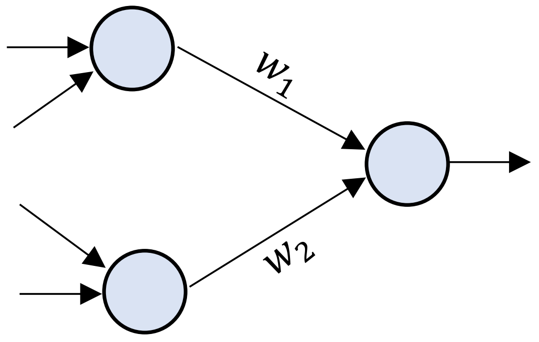


Figure 2.11: Synaptic weights between one layer containing 2 neurons, and an output layer.

training, the output of a given node does not change with a variable input sequence, then such a unit is not contributing to the solution and can be removed. Sietsma *et al.* showed that by pruning units, not only did the simpler network become faster, its accuracy and robustness to overfitting increased significantly. In the case of classification problems, pruning caused a 10-fold improvement in classifying noisy patterns with substantially simpler architectures [108]. The downside to this method was that the contribution of each node had to be individually evaluated.

Shortly after the introduction of pruning, LeCun *et al.* proposed an adapted version called *optimal brain damage* (OBD) [75]. The slight variation employed the second derivative when backpropagating the error through the network to compute the saliencies of each neuron. When applied, OBD managed to reduce the number of parameters in a practical neural network by a factor of four, while maintaining the same level of accuracy as pruning. Despite the success of this method, there is not enough emphasis on the fact that their program requires numerically evaluating and inverting the Hessian matrix, which contains the numerical second derivatives and is a significant computational expense. This problem was deemed circumvented in 1993 by Hassibi *et al.* who introduced *optimal brain surgeon* (OBS), which was proven to be more effective than pruning and OBD [48].

The accuracy of these methods stemmed from their reliance on computing the second derivative during backpropagation. However, as the applications became more complex, the computational costs started to increase greatly. Dropout built on the concept of pruning and introduced a retention probability into the network: each neuron is initiated with a random probability from a Bernoulli distribution that scales the weights during training. After a certain number of epochs, some nodes will be dropped out due to their retention probability dropping below a threshold. The resulting network is one that is far simpler than the original as entire layers have been automatically deleted in most cases.

Dropout has quickly become the most powerful tool for preventing overfitting and improving the generalization capability of a neural network. It has also shown to be equally effective across applications such as speech recognition, image detection and classification problems. The most effective version of Dropout to date includes a tailored activation function known as *Maxout* [41]. An in-depth introduction to Dropout may be found in [113] and a thorough summary is presented in [112]. More specific applications and versions of Dropout can be found in [24, 55, 109, 122, 123].

The most important conclusion from the extensive use of Dropout is that a neural network does not have to have a complex architecture to model complex problems, which is an ongoing misconception when building a neural network. Dropout can assist in automatically altering a network's architecture by removing units and layers (the opposite concept of adding neurons and layers has yet to prove its effectiveness), which simplifies the hyperparameter optimization process: one can simply start with an overly large network and let Dropout simplify it. The only drawback from Dropout is that the training process is typically 2 to 3 times longer [113].

2.3.5. Momentum

Neural networks with many hidden layers often face training difficulties due to the manner in which gradients are backpropagated from layer to layer. One common problem is known as the *vanishing gradient*. For example, backpropagating through a network with Sigmoid activation will induce a numerical update to each synaptic weight of 0.25 or less [5]. After moving through r layers, this value becomes 0.25^r meaning that after moving through 10 hidden layers, the update magnitude drops to 10^{-6} of its original value. At the other extreme, if activation functions with larger gradients are used or the weights are initialized at larger values, the gradient would increase to orders of magnitude higher than its original value. This is intuitively called the *exploding gradient*.

Such gradient problems have been more prevalent when using continuous activation functions. This is why piecewise numerical functions (Hard Tanh, ReLU, Maxout, etc.) have become so popular in recent years as they manage to circumvent the problem (the only exception is Google's Swish which is essentially a continuous representation of a numerical function). Both the exploding and vanishing gradients problems are equally bad as they hamper the network's ability to learn and converge to a solution. One effective workaround is to use Dropout to rid the network of useless neurons as explained in the previous section [48, 75, 113]. However, from a programmer's point of view, building a network with Dropout is far more complex than tuning the backpropagation scheme using a new parameter: *momentum*.

Momentum-based learning incorporates two new parameters into the network: the learning rate α and friction parameter $\beta \in (0,1)$. The former can be expressed by the initial decay rate α_0, epoch t, and variable decay rate k. The reason for using a learning rate is that it prevents a constant learning rate and training inconsistencies due to vanishing or exploding gradient issues. The two most used forms of learning rate are exponential and inverse decay [5, 49]:

$$\begin{aligned} \alpha_t &= \alpha_0 e^{(-k\cdot t)} \quad \text{[Exponential Decay]} \\ \alpha_t &= \frac{\alpha_0}{1+k\cdot t} \quad \text{[Inverse Decay]} \end{aligned} \tag{2.28}$$

The update scheme for the synaptic weights can now be expressed using an additional vector $\overline{V}$ which captures the exponential smoothing from the learning rate and friction parameters, shown below:

$$\begin{aligned} \overline{V} &\leftarrow \beta\overline{V} - \alpha\left(\frac{\partial L}{\partial \overline{W}}\right) \\ \overline{W} &\leftarrow \overline{W} + \overline{V} \end{aligned} \tag{2.29}$$

A slight adaptation of the previous scheme is known as Nesterov momentum and was first introduced in 1983. It showed that the converge error reduced to $O(k^{-2})$ after k steps rather than $O(k^{-1})$ in the case of traditional momentum. The altered updated scheme incorporates the friction parameter within the computation of the gradient to propagate additional information regarding how the gradients change [89]:

$$\begin{aligned} \overline{V} &\leftarrow \beta\overline{V} - \alpha\left(\frac{\partial L(\overline{W}+\beta\overline{V})}{\partial \overline{W}}\right) \\ \overline{W} &\leftarrow \overline{W} + \overline{V} \end{aligned} \tag{2.30}$$

There are many more adaptations of momentum-based learning all of which have been shown to be effective in certain applications such as AdaGrad, RMSProp, AdaDelta, etc. A complete list and explanation of all these variations can be found in [5]. However, a recent update algorithm that has become immensely popular was introduced in 2015 and is known as ADAM [61]. It self-adjusts its learning rate using a bias correction factor to account for unrealistic initialization of exponential smoothing parameters A_i and F_i. The former is the exponentially averaged value of the ith synaptic weight w_i. In most momentum schemes, A_i is the aggregate ith value:

$$A_i \leftarrow A_i + \left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i \tag{2.31}$$

In the case of ADAM, instead of using just the squared gradients to estimate A_i, exponential averaging uses the same decay rate k as in Equation 2.28. The reason behind this is that exponential averaging does not risk prematurely slowing down due to a constant scaling factor A_i, as in the case of simply using the aggregate values. In addition to A_i, ADAM exponentially smooths the first-order gradient through F_i with a different decay parameter β. The purpose of this is to incorporate momentum into the update process. The update scheme for ADAM can therefore be written as:

$$\begin{aligned} A_i &\leftarrow kA_i + (1-k)\left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i \\ F_i &\leftarrow \beta F_i + (1-\beta)\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i \\ w_i &\leftarrow w_i - \alpha_t \sqrt{\frac{F_i^2}{A_i+\epsilon}} \quad \forall i \end{aligned} \tag{2.32}$$

The inclusion of ϵ in the denominator is a small positive value such as 10^{-8} that allows for better conditioning of the update scheme [61]. One important note is that the learning rate α_t is different than before to incorporate the adjusted bias:

$$\alpha_t = \alpha \underbrace{\left(\frac{\sqrt{1-k^t}}{1-\beta^t}\right)}_{\text{Adjusted Bias}} \tag{2.33}$$

ADAM has shown to be one of the most effective kinds of momentum-based learning processes and is implemented in most neural networks regardless of their application. Training a network using momentum can significantly improve the convergence speed and the overall generalization capability of a network. However, Sutskever *et al.* pointed out that momentum is only beneficial in the case of a properly initialized network [115]. A network whose weights are initialized randomly might not benefit at all and could even be damaged from momentum-based learning [49, 115]. The key to enabling momentum-based learning is to perform pretraining on the network to initialize properly the synaptic weights.

Optimizing numerical convergence is not a recent topic and is extensively used when training a neural network. In all programming environments (TensorFlow, Octave, etc.), there are preset functions that automatically embed momentum into the training process. Regardless of application, momentum-based learning has become (almost) universally used in neural network development.

2.3.6. Additional noteworthy topics

The previous sections focused on a handful of advanced concepts in neural network design that are starting to be used more and more for practical uses. There are however a few concepts that focus on very precise issues which are important to be aware of even if they are not as widely used as the previous ones.

Data sparsity

Sometimes, a user is required to build a neural network capable of capturing an extremely complex problem, but only a limited amount of training data is available. Often, data sparsity occurs when conducting field work that involves capturing data from non-numerical phenomena. In the context of engineering, this might occur in laboratory experiments which introduce a significant amount of human and systematic error. An effective way for modelling a complex problem with sparse datasets is by using autoencoders such as that in Figure 2.10. By leveraging dimensionality reduction and expansion on the dataset, neural networks are effective and capable of generalizing problems well with limited amounts of training data.

A complete overview of sparse autoencoders can be found in Andrew Ng's Lecture Notes [91]. The literature also provides specific examples of applications of handling data sparsity in the field of speech and image recognition [25, 72–74], as well as in others [5, 49, 50]. Thankfully, the problem of sparse datasets will not come into play when building neural networks into finite element models since an unlimited amount of training data can be generated. In practice however, time constraints will come into play preventing the construction of infinite datasets.

Bagging and subsampling

On the other extreme of a sparse dataset is one that is infinitely large. This poses a unique problem in that the variance of such a dataset asymptotically reduces to 0 [49]. Variance reduction is desired as it can be the primary cause of overfitting, yet the problem stems from the fact that it is impossible to utilize an infinite database during training as it would simply take too long. Bagging is one method that is used to randomly sample the training data with replacement. The sampling size s is used to train a given model. Once optimized, the data are resampled with replacement to train another model. The resampled data will contain duplicates of the original dataset with a fraction of $(1-1/n)^n \approx e^{-1}$ where n is the size of the original training set. The outputs of the various models are then averaged to yield an accurate prediction [5]. Evidently, the disadvantage of this method is that it requires multiple training models which is highly time-consuming. However, the results show a system that has an exceptionally low variance and strong generalization capability. If parallel processing is available, then bagging is highly recommended.

Subsampling is almost identical to bagging. The difference is that subsampling samples the data without replacement. It has been shown that bagging performs better with a finite amount of data, however, when training data can be endlessly created, subsampling is more effective. In both cases, the resultant variance is far lower than simply training a network on a singular dataset permitting, better generalization.

Bagging first appeared in 1996 and was introduced by Breiman as *bootstrap aggregating* [20]. A complete mathematical analysis of bagging was published in 2002 by Bühlmann *et al.* [22]. A detailed study of bagging and subsampling within the context of neural network design may be found in [105, 131].

Boosting

Boosting is the opposite of bagging and subsampling. Whereas the latter two focus on variance reduction, boosting is a bias-reduction method that is typically applied to datasets with low variance and a high bias [7]. Boosting is most commonly used for classifiers and was first introduced in 1990 by Robert Schapire and Yoav Freund who proposed that weak hypotheses could be used to create highly accurate neural networks [103, 104]. Michael Perrone then extended on Schapire's work and proposed alternate uses for boosting in neural networks, notably regression problems [98].

In essence, boosting focuses on altering the weights within the training dataset, which is an entirely new concept. In a neural network architecture, the emphasis is on synaptic weights, however boosting also considers the importance of each data point in the training set. Figure 2.12 shows a

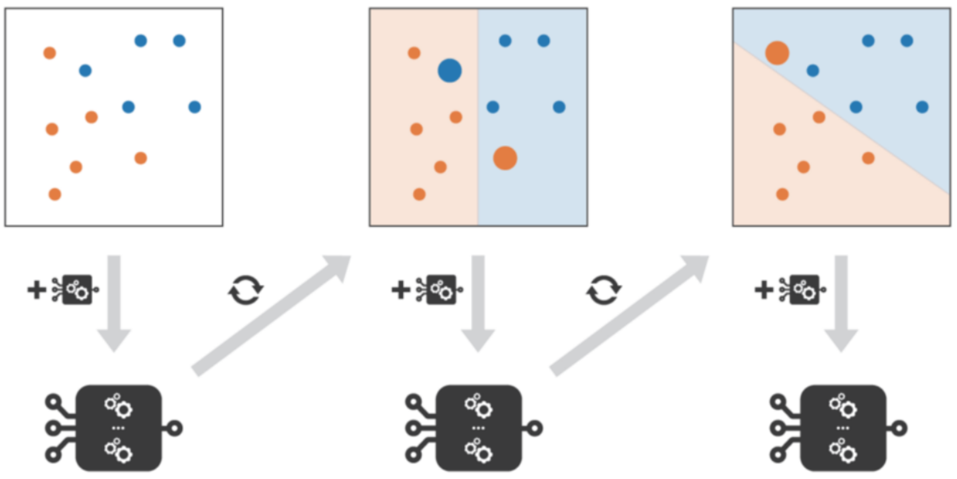


Figure 2.12: Example of a boosting scheme using AdaBoost [7].

brief overview of a boosting process. First the training set is used to train a shallow neural network. In the case that no backpropagation is used, the measured output error is directly related back to the training data which are reformatted to compensate for their error. In Figure 2.12 this is graphically shown by some data points becoming larger/smaller than others. This process is repeated until the resulting error converges below a predefined threshold. Such a process is evidently extremely powerful for classifiers to the extent that it is considered faster than training a neural network using backpropagation. However, from the moment the complexity of the problem increases (for example speech and image recognition or multi-output optimizers), then neural networks outperform ensemble schemes using boosting [103].

Backpropagation can also be incorporated into boosting but then the optimization process becomes difficult as it is very difficult to determine the optimal update scheme between updating the synaptic weights and altering the weights of the training set. AdaBoost and gradient boosting are two of the most common boosting techniques [5, 7, 103].

2.4. Closure

The goal of this chapter was to provide the reader with a sufficient background in neural network theory to understand how neural networks are used in practice, and some of the techniques that will be used throughout this thesis. The components of a neural architecture and training through backpropagation are unequivocally the most important concepts in neural network design. Additionally, it is very important to consider the following:

1. One of the biggest misconceptions in neural network design is that the more complex its architecture, the better it will be at generalization. Properly initializing the synaptic weights and the manner in which they are trained are far more important. This has been proven experimentally and theoretically by Bartlett in 1998 [9]. The result of Dropout also independently proves this notion by simplifying overly complex architectures to extremely simple ones that are just as expressive.

2. Overfitting is still one of the biggest problems, regardless of the application of the neural network. The quickest and most effective workaround is to deliberately inject noise into the dataset, which helps the network develop a sense of ignorance towards noisy patterns in the data thereby making it a better predictor. This has shown to be a difficult task for program-

mers to incorporate, since it is unnatural to deliberately introduce error into a system when it could be avoided [130].

All of the advanced approaches, with the exception of momentum-based learning, are seldom used in engineering applications. Moving forward, this presents a unique opportunity: incorporating these advanced concepts in network design for complex engineering problems such as finite element modelling could be an area of untapped potential.

3
Neural Network Deployment in FEA

Neural networks are known for their data-driven ability to accurately and efficiently model complex problems. They are therefore well suited for the field of engineering optimization where precise solutions are required from multivariable nonlinear problems. However, one interesting observation is that the employment of neural networks in such engineering disciplines is far less mature than in audio or speech recognition (some of the reasons for this will be put forward in Chapter 13). Nevertheless neural networks and machine learning are progressively entering the workplace. For example, NASA's Jet Propulsion Laboratory recently collaborated with Autodesk to create a machine learning interface that designed the structure of an interplanetary lander. From the governing boundary conditions, the resulting structural concepts all boasted organic-inspired models, which ventures far out of the conventional design space. Another example is that Airbus Defence & Space is starting to implement convolutional neural networks to accelerate the damage evaluation of solar arrays, thereby streamlining the maintenance process.

Finite element analysis is a tool that is widely used by engineers to model complex structures and materials under specific loading conditions (Zienkiewicz *et al.* have compiled a complete and in-depth study of the finite element method, which may be found in [132]). The results of such simulations are used in tandem with experimental tests to design efficient structures for tailored applications. Due to the heavy computational expense of finite element models, engineers are often faced with trading off accuracy for simulation time which can quickly become of the order of days depending on the model's complexity. Given that neural networks are purely data-driven systems, they would be able to theoretically improve the computational efficiency of a finite element model without compromising its accuracy. However, FEM models are extensive and consist of multiple strata of complexity making it difficult to know where a neural network might be effective.

This chapter provides a literature review to establish the state of the art of neural networks being employed in numerical structural optimization problems. Neural network usage in the context of FEA can be grouped into two broad areas: coupled and uncoupled networks. The distinction refers to the integration between the neural network and the FEA model. In an uncoupled network, the neural network works in isolation of the FEA model. In contrast, in a coupled network, the neural network resides within the FEA model and is fully *embedded.* Recall in Chapter 1 that research question RQ2 explicitly asks: "Is it feasible to embed a trained neural network into an active finite element analysis". Clearly, the focus of the thesis is therefore in the realm of coupled networks. To understand how such coupled networks might operate, however, it is instructive also to review quickly how an uncoupled network operates. Some of the literature on uncoupled networks, for example, has already addressed to some degree issues that this thesis seeks to address in a coupled network: structural failure and damage prediction; quick network training in a manufacturing context; post-buckling behaviour of panels or beams; complex crash behaviour and interpretation. Each of these issues in the uncoupled neural networks provide insights into how active integration through a coupled network can operate.

The chapter therefore commences in Section 3.1 with a quick review of uncoupled neural networks and their applications in damage prediction, manufacturing, and post-buckling, and crash behaviour. Section 3.2 builds on this and goes into further detail on the literature addressing active FEA integration in a coupled neural network. Section 3.3 provides a comparative discussion on how a coupled neural network may expand on or complement analyses currently relying on uncoupled neural networks.

3.1. Uncoupled neural networks

Uncoupled networks represent a class of networks that follow a scheme summarized in Figure 3.1. First, training data are generated from user-defined finite element models. Data are then split into the training and testing data. The former are used to train the neural network. This step is iterated until the model is sufficiently optimized and then the test data are incorporated to validate the model. The entire process is repeated until the neural network has obtained the desired generalization and expressivity capabilities. Once the workflow is complete, the model can be deployed and used independently [65]. The concept of an uncoupled network refers to the fact that after deployment, the neural network will not be embedded within a finite element model. It is meant to function as an isolated system. This section provides an overview of some of the applications for which such a workflow can be used.

3.1.1. Top-level damage prediction

One application of uncoupled neural networks is damage prediction. The manner in which materials and structures damage is an extremely complex topic but essential when considering how a design might eventually fail. Composite laminates are particularly intricate due to the multiple modes in which they can damage (matrix cracking, delamination, fibre breakage, etc.). In most ap-

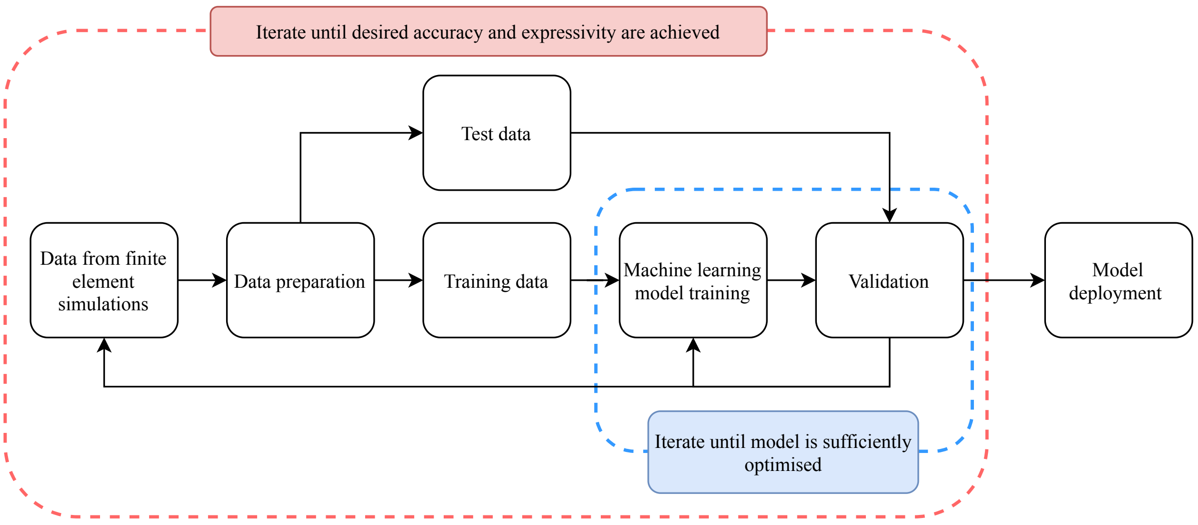

Iterate until desired accuracy and expressivity are achieved
Test data
Data from finite element simulations
Data preparation
Training data
Machine learning model training
Validation
Model deployment
Iterate until model is sufficiently optimised

Figure 3.1: Proposed workflow to train uncoupled finite element-based neural networks (recreated from [65]).

plications of damage modelling, neural network inputs are either the direction and magnitude of the loading vector, or strain values; and outputs are the location and size of the resultant damage. It is important to note that damage prediction can also be carried out using coupled neural networks, which will be discussed in another subsection. Top-level damage prediction refers to the type of inputs provided to the neural network. Often, training data take the form of top-level parameters such as the geometry and global loading situation of a structure. From these it is very difficult to obtain an accurate prediction of where the damage is going to occur, especially since the resultant network is meant to function in isolation.

One method for obtaining training data to model damage in composites is by using smart structures. These are composite structures with embedded sensors that provide real-time strain measurements. This is known as *active health monitoring.* One of the first examples of extracting such data to train a neural network appeared in 1992. Teboub *et al.* showed that once trained on a sufficient amount of experimental data, the neural network could be deployed as a self-contained program to provide quick diagnostics for the health of composite beams [117]. The resulting network was essentially a classifier capable of distinguishing between delamination, fibre breakage and matrix cracking based on an internal strain state provided by the sensors. Although the results suggested a potential for neural networks in such an application, the difficulty was in obtaining sufficient training data to be able to model all the conceivable kinds of damage patterns. The strain data provided by the smart structures were not enough to ensure a reliable classification. This was further emphasized by Labossière *et al.* in 1993 who predicted the failure envelope of a unidirectional composite laminate given the direction and magnitude of the loading vector [69]. Su *et al.* efficiently predicted delamination occurrence in glass fibre-reinforced laminates using fibre Bragg grating sensors [114]. Recent work by Elenchezhian *et al.* showed that the reliability of damage classifiers for composite structures could be drastically improved with modern non-destructive testing technology, particularly broadband dielectric spectroscopy [33].

Training neural networks on experimental data is a difficult task given the amount of human error and noise that is created, especially in the case of composite laminates. Finite element analyses can provide far more training data and are subject to fewer sources of error. Kudva *et al.* proposed an approach where a neural network was capable of predicting the damage size and location of a grid-stiffened aluminum panel under unidirectional compression, trained using finite element simulations [66]. For composite structures, leveraging the limitless data creating-capabilities of finite element analysis has shown to be particularly effective when only considering one mode of damage. Malik *et al.* conducted 100 simulations of composite plates being impacted by a rigid spherical solid travelling at $6.0 m/s$. The two-dimensional geometry was fixed, and the number of plies and stacking sequence was varied for both CFRP and GFRP. The resulting network was designed to predict the absorbed energy given a certain stacking sequence and material properties (CFRP or GFRP). The results showed a 99.91% correlation with test data while only using a network with one hidden layer [84].

The ability to capture or predict damage patterns using neural networks is attractive due to the heavy computational costs of modelling composite structures. Past research suggests that modelling using neural networks is most effective when using large datasets and focusing on just one mode of damage initiation. A classifier algorithm capable of predicting the type and location of damage given a certain geometry, loading conditions and stacking sequence is very appealing, but also too complicated to train. More examples of neural networks being used to predict delamination may be found in [23, 51, 120]. Alternative applications of uncoupled neural networks for damage prediction similar to those mentioned in this subsection may be found in [35, 77, 78, 83, 85].

An alternate approach for damage prediction relies on using coupled neural networks. Embedding machine learning algorithms at the material level within an active FEM analysis can overcome the lack of accuracy from top-level damage predictors generated by uncoupled neural networks. Nevertheless, the latter do enable a quick assessment of the type of damage and its location, although this should always be examined further due to the amount of uncertainty in such neural network models.

3.1.2. Manufacturing

Neural network-based optimization has also proven to be effective in the context of manufacturing. For instance, Shahani *et al.* applied it to a hot rolling process, which is one of the most common manufacturing processes since it concerns 80% of all metallic products, irrespective of their application [106]. Therefore, there is a significant incentive to optimize the process as much as possible. A finite element model was developed to output the stress, strain, strain rate, temperature and rolling force distributions for a set of input parameters such as thickness reduction, rolling speed, initial temperature, and friction coefficient. These were used to train a neural network with two hidden layers, which showed an excellent coherence with testing data [106]. Just as in previous cases, the resulting model was deployed as a self-contained, computationally efficient alternative to a full finite element model permitting the user to quickly obtain predictions regarding a hot rolling process. Such a tool is ideal for lab technicians working directly on the manufacturing line. A similar approach was used to optimize the process of air-bending forming of sheet metal by Fu *et al.* in 2010 [37].

3.1.3. Post-buckling and crash behaviour

Post-buckling represents an extremely nonlinear deformation regime of a structure. Modelling such a phenomenon proves to be a very expensive computational endeavor. Bisagni *et al.* proposed the use of neural networks and genetic algorithms to model the post-buckling behaviour of composite stiffened panels [17]. (Genetic algorithms are another machine learning tool, which are essentially a numerical manifestation of biological evolution. A practical introduction to genetic algorithms can be found in [28]). Instead of a singular neural network, a total of 10 neural networks were designed to work in parallel. The reasoning behind this was that it would be simpler to create multiple neural networks to model each type of structural response (buckling load, collapse load, etc.) rather than one cohesive network. Although this is debatable since a neural network that is properly calibrated could theoretically model a combined structural response, it is an effective approach given that the neural architectures were limited to two hidden layers and no more than 5 nodes in each layer. An added benefit of a clustered-network approach is that the training set for each network becomes small compared to that which would be required to train a cohesive system. The resulting scheme resulted in an average error of less than 10% across all types of structural response [17].

Recently in 2018, Abambres *et al.* published a study modelling the post-buckling behaviour of steel beams using a singular neural network. Using similar inputs as Bisagni *et al.*, Abambres managed to further reduce the average error to 2%, thereby proving that it is possible to model the combined structural response of a post-buckled beam with one neural network [3]. However, it should be noted these improvements were conducted on a steel beam, which is easier to model than a composite beam as in the case of [17].

Lanzi *et al.* used the same approach as in [17] (multiple small neural networks working in parallel) to reproduce load-time curves of structural behaviour during the event of a helicopter crash [70]. Finite element analysis using PAMCRASH [1] was used to generate the training data that consisted of the scaled design variables as inputs, and consequently returned the required components to generate a load-time curve (load at first peak, maximum force, and mean force). Again, a system of small neural networks running in parallel proved to be effective as the resultant average error was 12% [70]. It is important to emphasize that creating a system of clustered neural networks experiences diminishing returns as the complexity of the problem increases, as the global training process will become less efficient than that of a singular neural network. Furthermore, it also becomes far more difficult to troubleshoot discrepancies and improve overfitting, since the loss function in the final combined output cannot be traced back to the responsible neural network(s).

3.2. Coupled neural networks: active FEA reintegration

Coupled neural networks rely on a slightly different workflow structure than uncoupled networks. In the previous examples, the resulting networks were deployed for self-contained usage, i.e., once they were trained, they were meant to function on their own. Given that they were designed for a specific application, their data-driven architecture could easily bypass the computational expense of complicated finite element tasks. Coupled networks take the trained networks one step further by reintegrating them back into an active finite element model. This new workflow may be seen in Figure 3.2. Depending on how they were trained, multiple units of the same neural networks could be reintegrated to work together along with the rest of the computational model to accelerate the entire finite element analysis. The application of the neural networks could take any form of computational task that is performed over the course of a FEM analysis, thereby improving computational efficiency at key locations. Theoretically, this would allow the user to conduct an accelerated finite element simulation on any given application, without compromising accuracy.

One example of coupled neural networks was developed by Liao *et al.* who embedded active neural networks within a finite element simulation of a waveguide filter to accelerate the determination of spatial EM-field couplings [80]. Although their method is not evidently relatable to computational solid mechanics, it does show that an FEA can be substantially accelerated without losing accuracy by using trained neural networks. Liao *et al.* stress that the success of this method relies on a robust training scheme and a seamless understanding of the thresholds between neural networks and the remaining numerical tasks. Another case was proposed in 2005 by Ramuhalli *et al.* who integrated neural networks to determine the global stiffness matrix of a mesh using the material properties of each element as inputs [100]. The computational efficiency of their model heavily hinges on constant boundary conditions which were used to train the network. The purpose of their model was to solve complex differential equations for electromagnetic nondestructive evaluation.

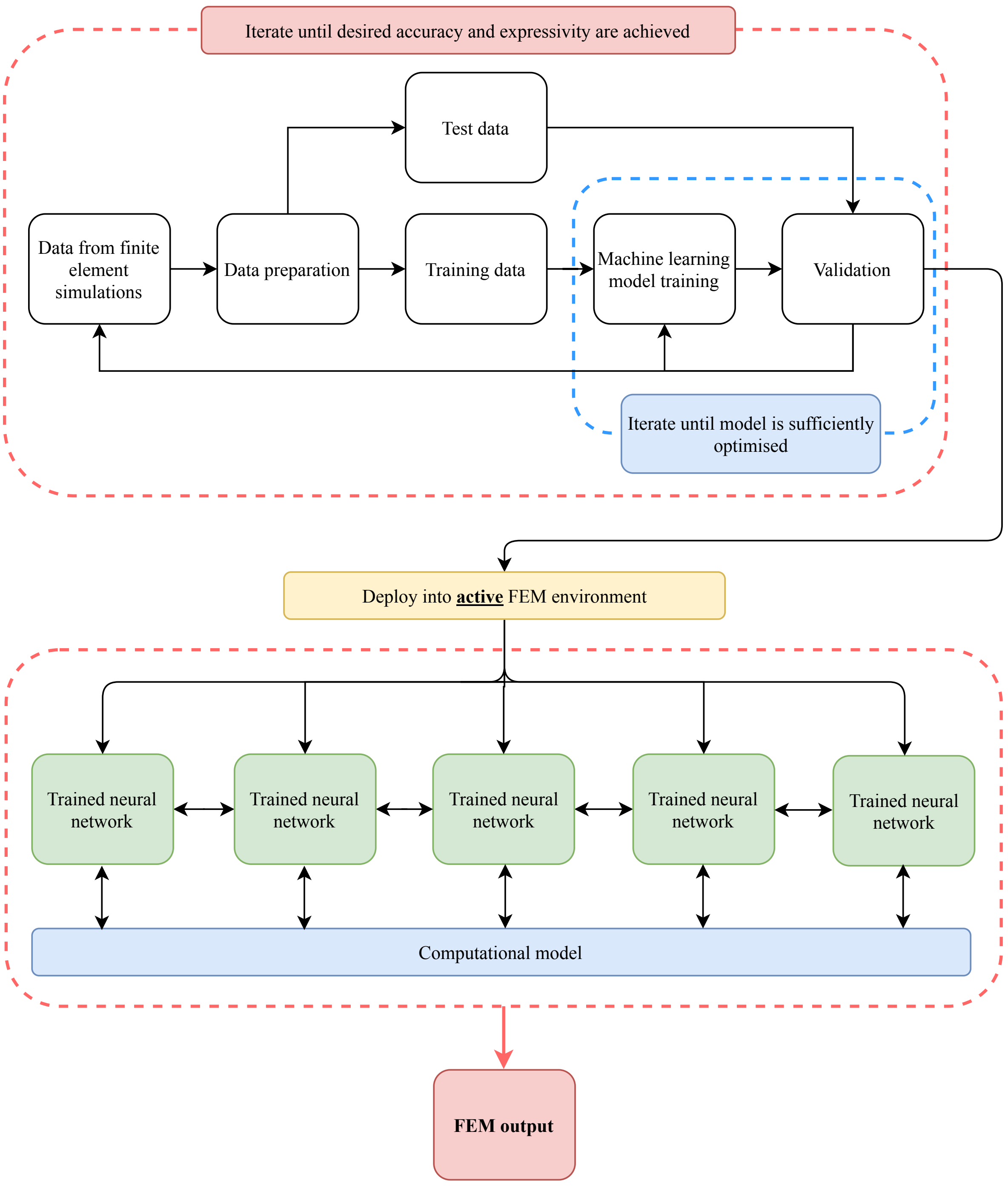

Iterate until desired accuracy and expressivity are achieved
Test data
Data from finite element simulations
Data preparation
Training data
Machine learning model training
Validation
Iterate until model is sufficiently optimised
Deploy into **active** FEM environment
Trained neural network
Trained neural network
Trained neural network
Trained neural network
Trained neural network
Computational model
FEM output

Figure 3.2: Modified workflow for active FEA reintegration.

3.2.1. Constitutive modelling

Within the context of structural analysis, one area of research pertaining to coupled neural networks is constitutive modelling. Stress-strain relations, also known as *constitutive relations*, intuitively relate the stress and strain of a material to model its mechanical response. For example, in the case of a linear elastic material, a constitutive relation would be:

$$\begin{aligned} &\sigma = \mathbf{D}(\varepsilon - \varepsilon_0) + \sigma_0 \\ &\textit{or:} \\ &\varepsilon = \mathbf{D}^{-1}(\sigma - \sigma_0) + \varepsilon_0 \end{aligned} \tag{3.1}$$

Where **D** represents the material matrix. Constitutive models are used in finite element analyses to define the behaviour of an element and can be embedded in an active FEA at the material level via a user material subroutine in ABAQUS (UMAT). Depending on the complexity of the material, multiple constitutive models might be used to capture more complex mechanical responses such as plasticity or fracture. Neural networks were first introduced in the 1990s by Ghaboussi *et al.* as an alternate, data-driven approach to building a constitutive model [38]. The networks should model a constitutive relation, its inputs and outputs should be the stresses and strains respectively. However, Ghaboussi identified that the major difficulty would be capturing the nonlinear mechanical response since material behaviour is path-dependent. The solution was to develop a stress-controlled model: the inputs of the network would be the current states of stress and strain $(\sigma_1, \sigma_2, \varepsilon_1, \varepsilon_2)$, as well as the stress increment $(\Delta\sigma_1, \Delta\sigma_2)$. The resulting output would be the predicted strain, i.e. $\Delta\varepsilon_1$ and $\Delta\varepsilon_2$. A neural network consisting of two hidden layers with 40 nodes in each layer was developed for the constitutive model, which was trained on experimental data extracted from biaxially loaded concrete.

Figure 3.3 shows the neural architecture that was used by Ghaboussi *et al.* to develop a constitutive model. Once trained, the network was deployed to predicted the constitutive relationship of biaxially loaded concrete. When compared with experimental data, it was found that the data-driven network could reasonably reproduce the nonlinear mechanical response of the material, which was a promising outcome for network-driven computational mechanics [38, 59]. It is however interesting to note that there was no motivation for the choice of neural architecture. It was simply determined by trial and error. In their paper, Ghaboussi *et al.* do stress that more research

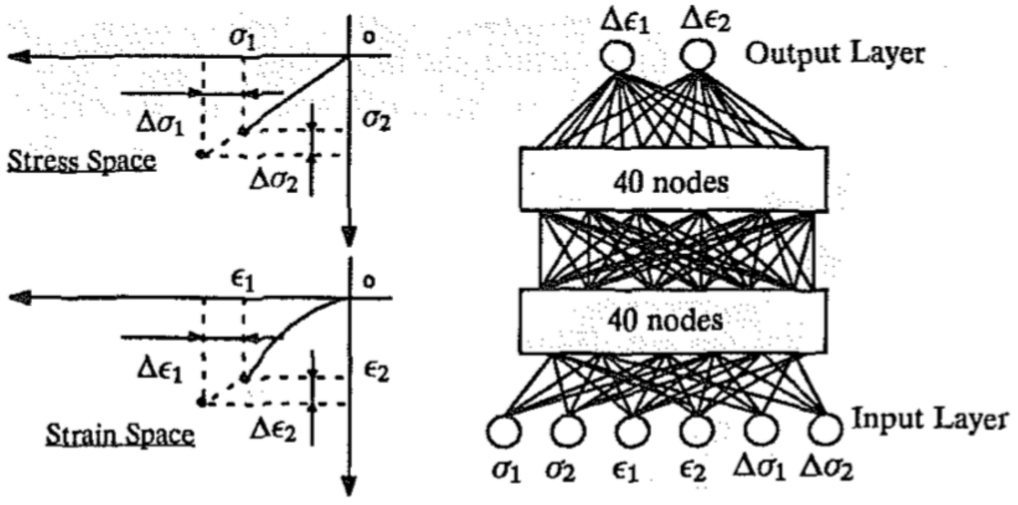


Figure 3.3: Neural architecture used for developing a constitutive model of biaxially loaded concrete [38].

should be invested in building a neural architecture for structural engineering applications. However, the paper incorrectly postulates that the discrepancies between the artificial neural network and the experimental data are due to a lack of complexity in the neural architecture. As was shown in [9], augmenting the complexity of a neural network does not necessarily increase its expressivity. Rather, more emphasis should be put on weight initialization. In the 1990s, a significant amount of research had already been conducted in the field of neural network design and it is surprising that it was not used when designing such experiments. Despite this, Ghaboussi *et al.* introduced the first promising application of a neural network in structural computational mechanics at the material level. Although some might argue that Kupfer *et al.* initiated the notion data-driven computational mechanics in 1969, no neural networks were applied at such stage [68].

Constitutive modelling using neural networks really started to gain momentum in 1998 [39], when they were used to model the behaviour of a composite laminate plate containing an open hole. Composite materials are widely used in aerospace engineering for their high specific properties. Unfortunately, the complexity of a composite laminate cannot be overstated. Multiple plies, each containing fibers in specifically defined directions, carry, transmit and propagate the load differently depending on their respective position to various boundary conditions. Accurately modelling composite structures is widely responsible for high computational costs in finite element models. Hence, there is significant interest in rendering this process more efficient using alternate methods such as machine learning. Two concise yet in-depth overviews of numerically modelling composite materials can be found in [81, 116].

Ghaboussi *et al.* introduced a new approach to designing a neural network known as *autoprogressive training* [39]. This essentially starts with a simplified neural architecture, whose weights are first trained. Neurons are then added to each layer (note that no new hidden layers are inserted), the weights that have already been trained are frozen, and the new synaptic weights are trained on resampled data. Finally, the old connections are unfrozen and all the synaptic weights are fine-tuned together. This could be interpreted as a form of bagging but used within a singular model instead of across multiple networks, or it could also be considered as a kind of supervised pretraining. In

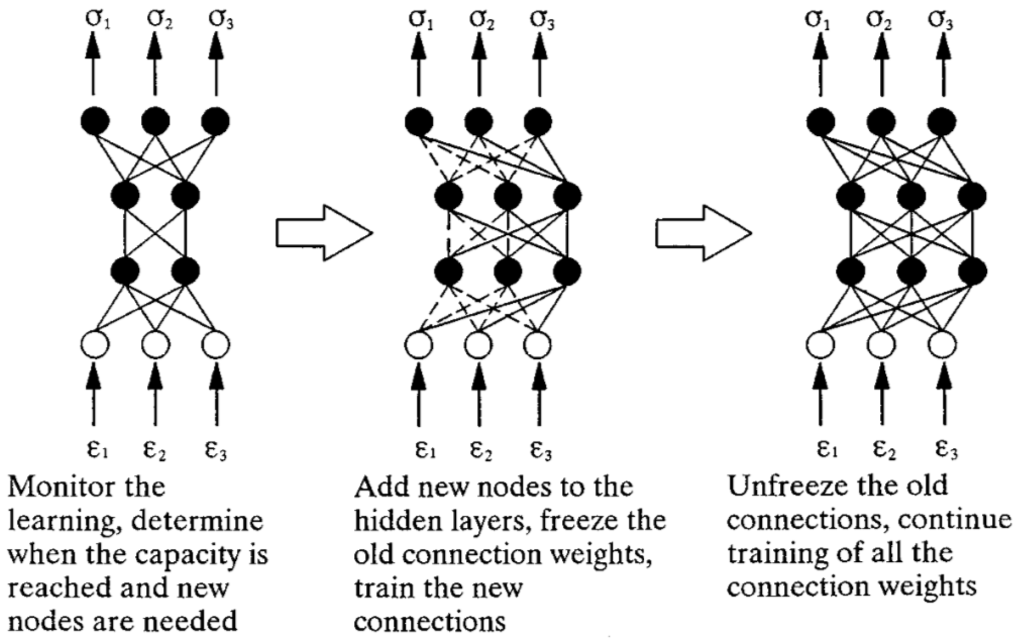
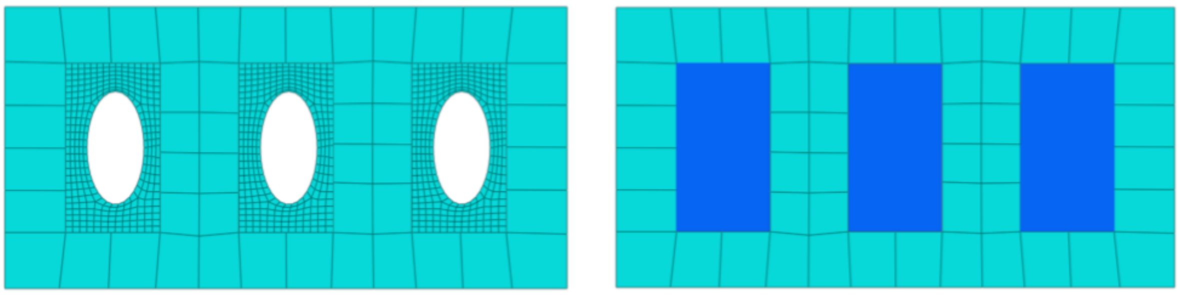

σ1 σ2 σ3
ε1 ε2 ε3
Monitor the learning, determine when the capacity is reached and new nodes are needed
Add new nodes to the hidden layers, freeze the old connection weights, train the new connections
Unfreeze the old connections, continue training of all the connection weights

Figure 3.4: Autoprogressive training overview [39].

any case, the end goal is the same: improve the generalization capability of the network by properly initializing the synaptic weights (thereby reducing the variance of the overall system). Figure 3.4 shows an overview of this process. Hashash *et al.* used this same framework to model the material behaviour of inelastic metal through an inverse shift analysis [47].

The autoprogressive training scheme was used to develop a constitutive model that predicted the stress increment from the strain state and strain increment. As opposed to the previous case where experimental data was used, Ghaboussi *et al.* used finite element analyses to create a vast training set for the network [39]. The training scheme was analogous to the workflow in Figure 3.1 and is shown in detail in Figure 3.6. A composite plate with an open hole under compression was used to test the neural network. Experimental data from strain gauges measuring the shortening of the plate were compared to the results from the neural network that was trained at the material level using a finite element model. The results were very promising showing that neural networks were also capable of modelling complex composite materials. Autoprogressive training was shown to be effective for initializing the synaptic weights, but did not overcome the lack of expressivity which was still present. A trial-and-error approach was used to determine the number of hidden layers: two. A review of neural networks being used to model fiber-reinforced polymeric composites may be found in [44].

Modelling the constitutive relations via neural networks has since been applied to more complex structures. Tom Gulikers leveraged their expressive capabilities to model a composite laminate with an elliptical hole under biaxial tension. Stress-strain relationships were extracted at the boundary and showed excellent coherence with test data generated from finite element simulations. The architecture of the neural network was determined by trial and error, and it was found that two hidden layers were sufficient to capture the mechanical response of composite structures, including their damage criteria [42]. This required reformulating the elasticity matrix to account for damage and ultimately produced satisfactory results while using a neural network with only two hidden layers. Gulikers took the devised framework one step further and postulated that a trained constitutional model could be reintegrated back into an active mesh [42]. For instance, if the neural network was trained on a plate with an elliptical cutout, the resulting model could be used to replace such cutouts in larger structures as shown in Figure 3.5. Unfortunately, such an approach is limited when complex non-uniform or non-periodic boundary conditions are applied on the RVE or substructure. Therefore, another approach should be employed; for instance: training a neural network at the element level rather than at the material-point level and then proceeding with mesh reintegration.



(a) Traditional mesh discretization. (b) Neural network constitutional model.

Figure 3.5: Modelling elliptical cutouts in a fuselage panel [42].

One final example that is slightly different than those already mentioned centers on human bone fatigue crack growth. Hambli *et al.* devised a hybrid framework to build neural networks at the integration point level to accelerate the transition from macro to meso scales by determining the crack length and density based on the cycle number, apparent stress, density, and Young's modulus [45]. The neural network was incorporated into ABAQUS using a UMAT subroutine. Figure 3.7 shows the devised framework by Hambli *et al.* Although a sufficient amount of experimental data is lacking to validate the cohesive model, the results were promising as well as a staggering $4.32 \cdot 10^5$ times faster than the original finite element simulations. Oishi *et al.* also devised a similar approach at the integration point level, but on a deeper level: the neural network was used to enhance numerical integration of the element's stiffness matrix using the Gauss-Legendre quadrature. The minimum number of integration points required for a given error tolerance was estimated using neural networks. Overall, this allows for each element to use the optimal number of integration points to reduce computational expense without sacrificing accuracy [94].

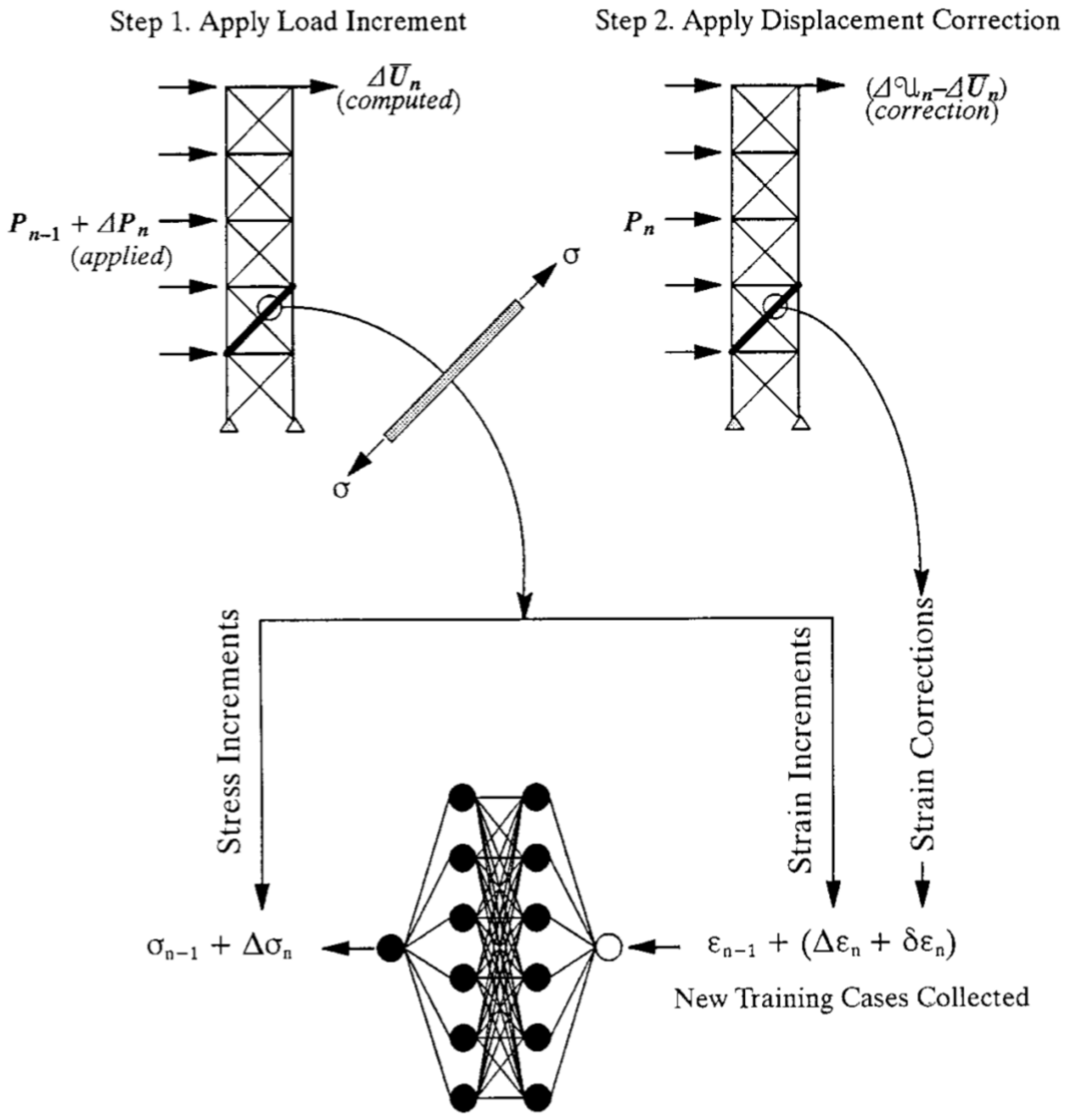

Step 1. Apply Load Increment
Step 2. Apply Displacement Correction
$\Delta \bar{U}_n$ (computed)
$(\Delta \mathcal{U}_n - \Delta \bar{U}_n)$ (correction)
$P_{n-1} + \Delta P_n$ (applied)
P_n
σ
σ
Stress Increments
Strain Increments
Strain Corrections
$\sigma_{n-1} + \Delta\sigma_n$
$\varepsilon_{n-1} + (\Delta\varepsilon_n + \delta\varepsilon_n)$
New Training Cases Collected

Figure 3.6: Neural network training to develop a constitutive model from finite element training data [39].

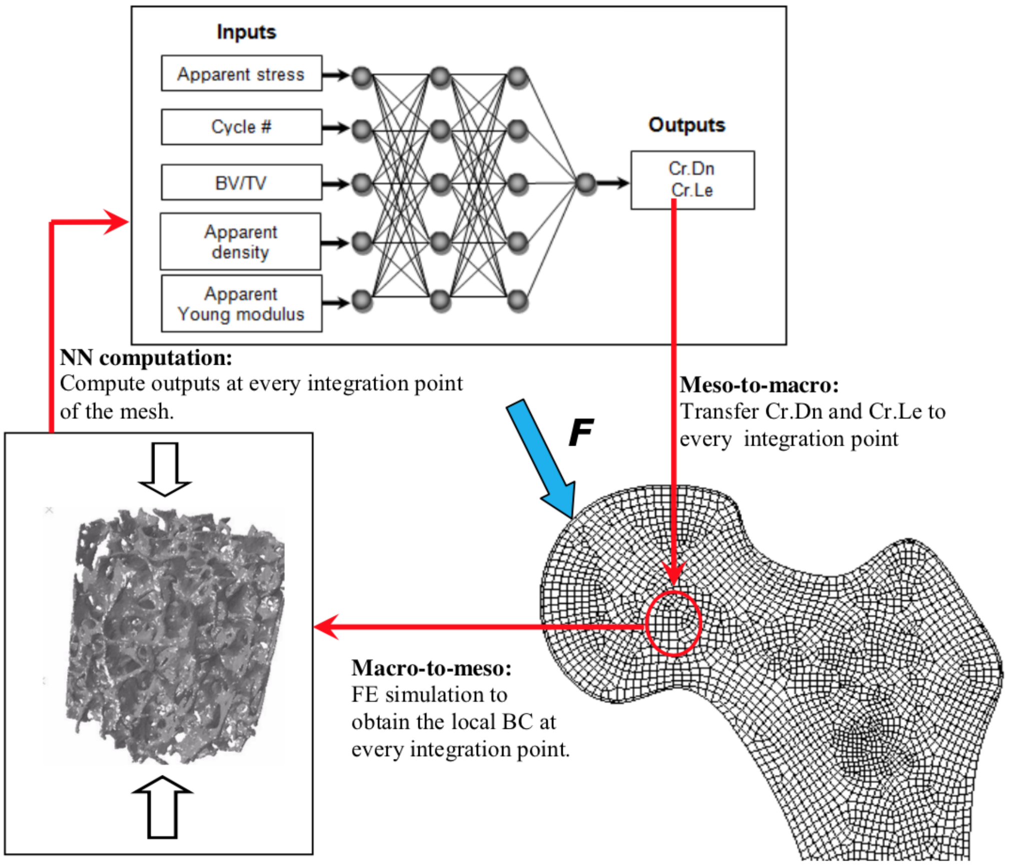

Inputs
Apparent stress
Cycle #
BV/TV
Apparent density
Apparent Young modulus
Outputs
Cr.Dn
Cr.Le
NN computation:
Compute outputs at every integration point of the mesh.
Meso-to-macro:
Transfer Cr.Dn and Cr.Le to every integration point
F
Macro-to-meso:
FE simulation to obtain the local BC at every integration point.

Figure 3.7: Multiscale finite element approach using neural networks embedded at the integration point-level [45].

Constitutional modelling represents one of the larger areas of coupled neural networks being used for structural analysis. Neural networks also proved capable of modelling cyclic behavior at a material level [39, 129]. More examples of constitutive modelling using neural networks may be found in [40, 46, 57].

3.2.2. Framework development & data management

The expressive power of neural networks has proven to be extremely successful when reintegrating back into an active finite element model, as shown in the previous section. This then prompts the desire to extend neural networks to more complex structural problems, at which point a new issue might come into play. For a given problem, the dimensionality of the design space is exponentially linked to the required number of sampling points. Bessa *et al.* described this as the *curse of dimensionality* [15], a concept that was first introduced in 1961 by Bellman when evaluating the effectiveness of grid search [10]. In essence, if the number of sampling points required to train a neural network is large enough, the use of a neural network might start to experience diminishing returns as the time required to train it is equal to or exceeds the time required to develop a traditional FEM model. Several recent findings have developed frameworks to manage the copious amounts of data in high-design space problems without impairing the effectiveness of data-driven solutions.

In 2016, Liu *et al.* developed a novel technique for organizing the data of problems with large sampling dimensionality called *self-consistent clustering analysis* [82]. The idea behind this was to help lower the size of the training domain by decomposing a high-fidelity Representative Unit Cell (RUC) into material clusters. Figure 3.8 shows an overview of the developed cluster analysis. The offline stage essentially subdivides a high-fidelity RUC into a reduced-order model which is then used on the online stage in the predictive process. The simplified model permits faster computations by reducing the dimensionality of the data into material clusters. Although in his example Liu did not use neural networks, the framework could be easily extended to the training domain of a network by using only data from key representative locations in a mesh rather than at every point.

Bessa *et al.* built on principles of the self-clustering analysis to develop a new framework that builds large databases that are suitable for machine learning shown in Figure 3.9 [15]. One key element in this process is the Design of Experiments stage (or DoE), which uses space-filling analysis to treat similar inputs as a cohesive unit and could be argued to be an extension of Liu *et al.* self-clustering analysis [82]. An overview of various types of space-filling methods was compiled in 2001 by Simpson *et al.* [110]. The purpose of the DoE stage is to identify the required design samples to build a representative yet efficient domain for computational analysis, which is also considered as the bottleneck of the framework. The design samples are then passed through computational analyses where the outputs are used to build machine learning models to make predictions on new models. The entire process can then be refined, starting by refining the sampling in the DoE stage. If the generated design samples prove to create machine learning models that are not sufficiently accurate, then the space-filling methods can be re-evaluated to increase the number of samples. On the other hand, if the accuracy of the machine learning model is adequate, the DoE process could be revised to group the data even more to promote increased computational efficiency. Bessa showed that the devised framework serves as a robust starting point for rendering complex structural problems with high-dimensionality design spaces accessible for data-driven solutions [15]. The key element remains the effectiveness of the space-filling methods in the DoE phase, which drastically reduces the number of required design samples.

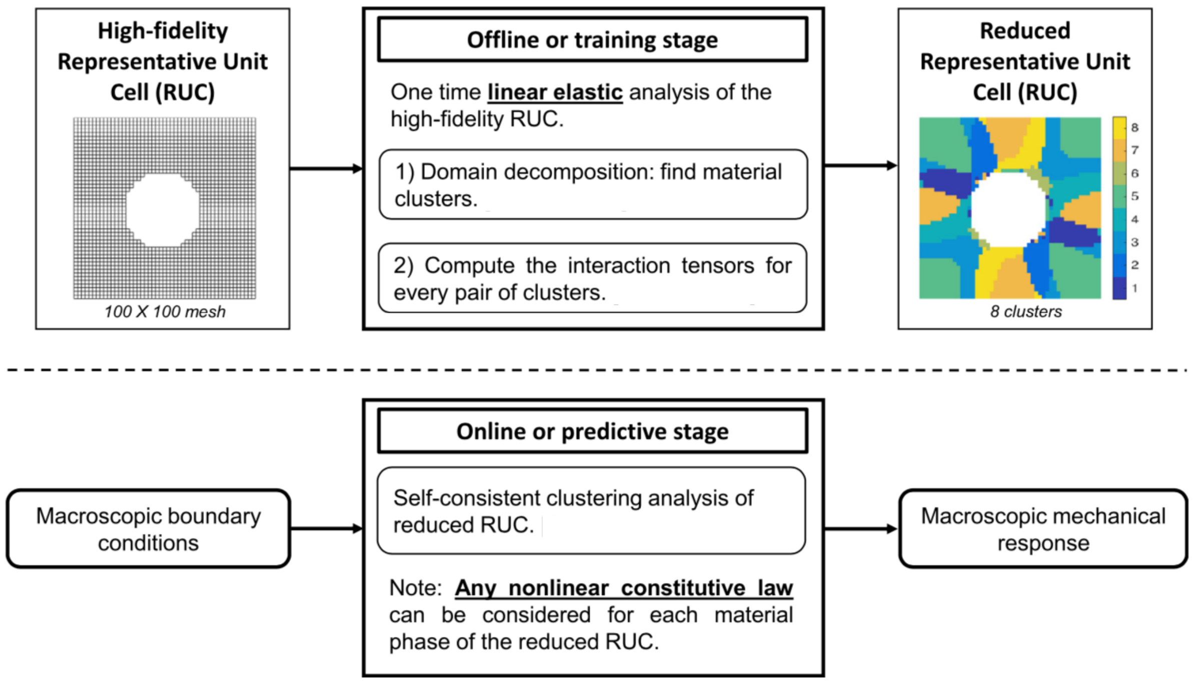

High-fidelity Representative Unit Cell (RUC)
100 X 100 mesh
Offline or training stage
One time linear elastic analysis of the high-fidelity RUC.
1) Domain decomposition: find material clusters.
2) Compute the interaction tensors for every pair of clusters.
Reduced Representative Unit Cell (RUC)
8 clusters
1
2
3
4
5
6
7
8
Online or predictive stage
Macroscopic boundary conditions
Self-consistent clustering analysis of reduced RUC.
Note: Any nonlinear constitutive law can be considered for each material phase of the reduced RUC.
Macroscopic mechanical response

Figure 3.8: Graphical overview of the self-clustering analysis [82].

Reducing the dimensionality of the sampling space for complex problems is essential when using neural networks or else the accuracy of the model will not outweigh the computational expense during training. Although the previous examples pertain to the micromechanical domain, they are just as applicable to higher-level problems. Typically genetic algorithms are used in high-dimensional optimization problems, however, they lack the expressive and predictive capabilities of neural networks. Hybrid approaches that combine a neural network with a genetic algorithm are sometimes used to help reduce the design samples for the neural network as shown by Bisagni *et al.* [17]. Although this was shown to be effective, it requires simultaneously developing two types of machine learning models. Bessa and Liu's framework, on the other hand, revolves around one machine learning model and focuses on the DoE phase to reduce the dimensionality of the problem [15, 82].

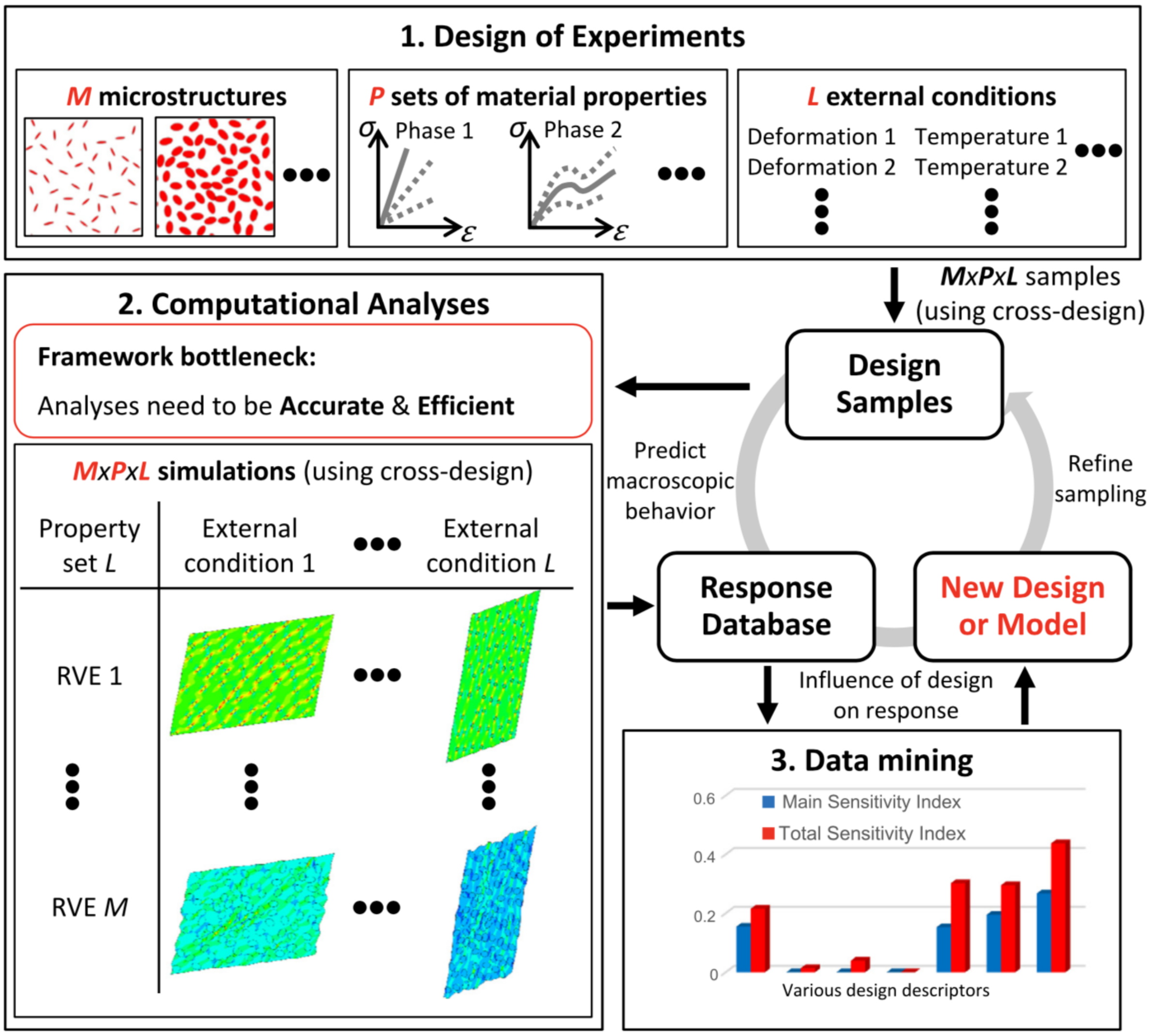


Figure 3.9: Overview of global framework developed by Bessa *et al.* [15].

3.3. Closure

As shown throughout the previous section, coupled neural networks are a relatively new area of research. This is mainly because it is very difficult to properly determine where neural networks should be embedded within active finite element analyses, especially in structural analysis applications. Compatibility with the other computational components such as the overall mesh and adjacent elements has to be seamless. Otherwise, the coupled neural network might worsen the computational expense of the model. Despite the challenges, integrating machine learning within finite element analyses appears to be an area of high potential for improving the efficiency of complex structural optimization problems.

Uncoupled neural networks, on the other hand, are being widely used for structural optimization tasks. Their self-contained state allows the user to quickly obtain accurate results concerning complex optimization tasks that would otherwise require extensive finite element models. It should be emphasized, however, that such models are only functional within an input parameter domain defined by the training set. Anything outside such a domain would require the network to extrapolate, which no longer guarantees the same degree of accuracy.

Concerning the neural architectures and learning schemes, it is surprising how little emphasis is put on network hardening given the current state of research in the field of machine learning. No study in the literature reviewed in this chapter conducted sensitivity testing on the network's performance or utilized any of the more advanced topics in neural network design to further improve model performance. All the neural architectures were formed by trial and error leading to a network design with two hidden layers in most cases and used a Levenberg-Marquardt training algorithm [43, 79, 86]. Despite its effectiveness, it has not shown to be as efficient as other momentum-based learning algorithms such as ADAM [61]. Only Ghaboussi *et al.* and Bisagni *et al.* showed creative and novel approaches to designing neural networks, using autoprogressive training and clustered neural networks respectively [17, 39].

The potential of integrating neural networks into engineering applications is immense. There is also an extended literature regarding data-driven approaches - other than neural networks - to model physics-based problems [26, 36, 62, 92]. One common factor between all the examples cited in this chapter is that the computational efficiency of structural simulations can be greatly improved without compromising accuracy. This alone is sufficient to further pursue this field of research, and see just how far the modelling capabilities of neural networks can be pushed. The literature presented in Chapter 2 (neural network design) and Chapter 3 (neural network use in FEA), provides the confidence and motivation to build a coupled neural network at an element-level for active usage during an FEA simulation. The research objective for this thesis presented in Chapter 1 along with the supporting research questions, is reiterated below.

The objective within the time-span of the thesis is to reduce the computational expense of a finite element analysis – without compromising its accuracy – by embedding neural networks trained on an element level into an active mesh.

4 Computational Architecture Design

Building a neural network at an element level to be used in active FEA simulations is a complex multi-platform endeavour requiring meticulous planning. This endeavour involves designing a custom finite element with a built-in neural network. The element can be imported into the FEA software by the user. Such an element is commonly referred to as a *user-element* (UEL) or *user-subroutine*. Because a goal of this thesis is to integrate an active neural network into the user-element, the UEL will also be referred to as a *neural element*. This chapter outlines the computational framework used to create and deploy such a neural element within an active FEA simulation.

4.1. The appropriate context: biomimicry

User-elements are typically developed when the conventional elements present in a finite element library do not satisfy the user's needs. For instance, modelling delamination in composites requires specially formulated elements that can take into account the regions of large plastic deformation and material yielding. Another example are geometrically-exact slender beam elements of arbitrary curvature, which undergo large displacements and rotations. However, within this research framework, a new element is created to improve the computational efficiency of elements that already exist using machine learning. The chosen context therefore should be as simple as possible, while still being able to demonstrate the capabilities of the neural element.

A logical starting point is a one-dimensional modelling space, where only axial loads are present. This intuitively corresponds to truss structures, where each truss member can only be loaded in either tension or compression. Creating a machine learning framework to model the deformation of simple truss structures could be argued to be excessive, since it is very straightforward to build such models using existing elements. However, it becomes appropriate when considering that the neural element could potentially be used to model complex truss structures and incorporate nonlinear deformations, which extend beyond the capabilities of traditional FEA truss elements.

Organic-inspired structures are becoming more widespread thanks to improvements in 3D-printing manufacturing. Topology optimizers are now fully present during the design stage in engineering projects. With the required boundary conditions, the optimizers are capable of designing potential structures that satisfy all the design requirements, while minimizing the overall mass. In most cases, the resulting structure has shown to be more lightweight and efficient when compared to traditional design methods. Figure 4.1 shows an example of a topology optimization scheme conducted on a bracket.

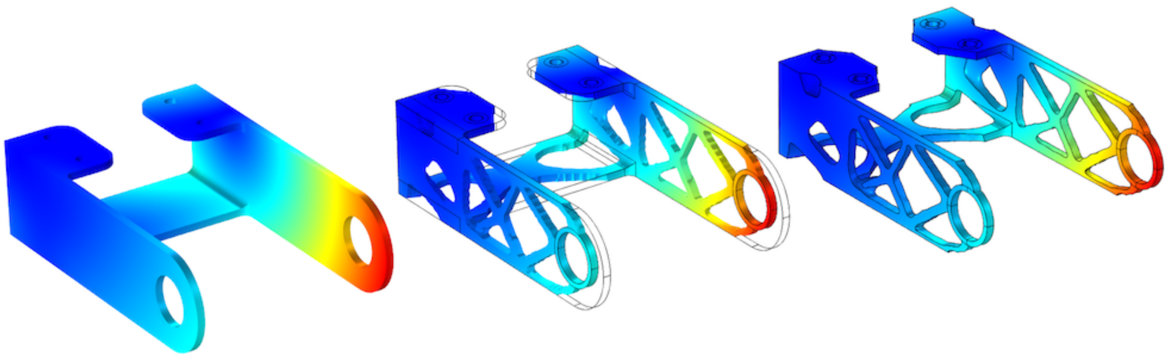


Figure 4.1: Example of a topology optimization process conducted on a bracket. The resultant structure (far right) is capable of withstanding the original loads and boundary conditions, but is lighter than the initial design (far left) [58].

However, the complexity of organic-inspired structures is a different matter. Their unique geometry is often referred to as *organic*, since they emulate structures found in nature such as structural veins within leaves (Figure 4.2) or those found in human shin bone (Figure 4.3). Both of these possess a multi-truss structure. Thousands of small structural members of varying shapes and sizes form an intertwined lattice of the final product: a lattice best manufactured using 3D-printing techniques. In addition to the manufacturing process of such structures, it is also difficult to predict accurately their mechanical response, since finite element models quickly become complex and computationally expensive. Additionally, predicting the material defects due to the 3D-printing process and how the defects affect structural integrity remains a persisting issue. These organic structures have opened an entirely new field of structural engineering known as *biomimicry* and is believed to be the future of structural design [8]. Unfortunately, as of now, organic structures are typically dismissed in the early design stages due to their complexity both in terms of manufacturing and computational modelling.

Despite their complexity, the lattices emerging from biomimicry can be simplified to a random collection of interconnected truss members, each loaded under either tension or compression. Consider the example in Figure 4.4. The structure on the far left depicts part of a multi-truss

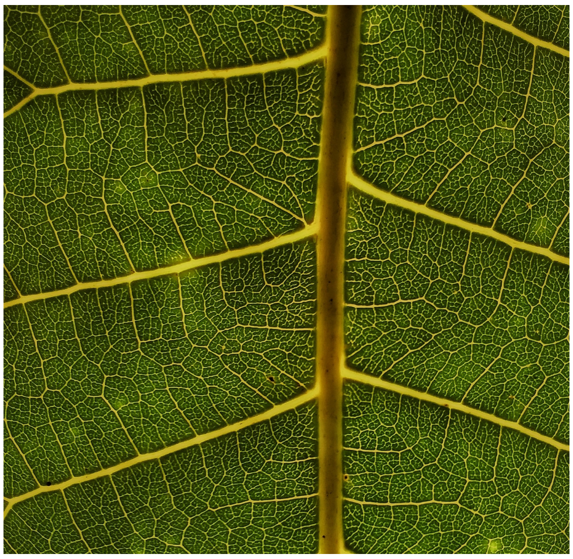


Figure 4.2: Organic structural lattice of a leaf [29].

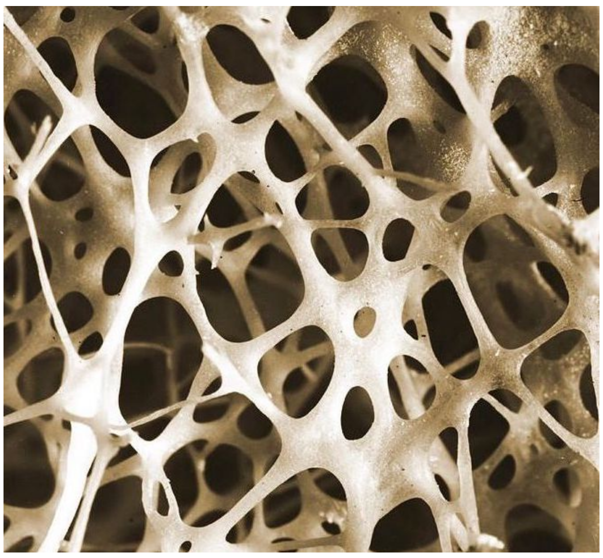


Figure 4.3: Porous bone structure [111].

lattice, which might be the outcome of a topology optimization process. The red nodes indicate the points where the truss members are interconnected. If this structure is loaded in any fashion, the loads propagated through each truss member will not be the same. Some will be under tension, and others under compression. If the deformation of each truss member remains within the linear-elastic domain, then the overall structure is readily modelled by replacing each truss member with a truss element in ABAQUS (T2D2). However, if the loading scheme on the structure is such that some truss members are compressively loaded, and if manufacturing defects are present, then these truss members might buckle and enter the post-buckling regime. Such a situation cannot be modelled using truss elements, since truss elements do not allow for deformation out of the axial plane. Instead, beam elements (B22 for example) would have to be used to model the buckling phenomenon. In addition, buckling and post-buckling regimes also require a finer mesh, therefore one beam element cannot be used to model one truss member. Instead, multiple beam elements are required for each truss member to achieve an accurate solution. An example of such a mesh discretization, shown in Figure 4.4, and naturally comes with a higher computational cost. Given that multi-truss structures resulting from biomimicry can be highly complex and possess multiple slender truss members, this creates an opportune context for a neural element. A neural element can be designed as a truss element to be used in multi-truss structures. It can be loaded axially while also incorporating the out-of-axial plane deformation found in beam elements; thereby improving computational efficiency. This would lead to a FEA model where each truss member could be replaced with a user-element as shown on the far right of Figure 4.4.

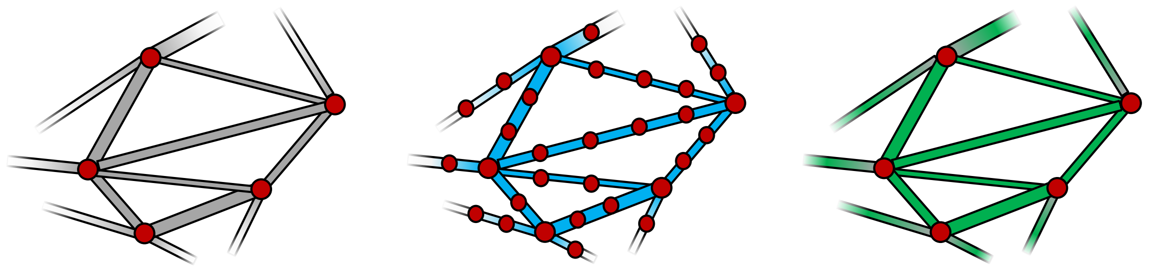


Figure 4.4: Example of the practical usage of a neural element. Far left: initial planar multi-truss structure. Middle: traditional FEA discretization (traditional FEA shown in blue). Right: discretization using neural elements (neural element shown in green).

For this thesis research, the neural element will be developed as a truss-like element that incorporates the modelling capabilities of beam elements. The final user-element will be able to deform under tension and compression, and include out-of-plane modelling capabilities such as buckling and post-buckling. To summarize, the full deformation field the neural element will encompass:

- Axial tensile deformation including material plasticity.
- Axial compressive deformation including post-buckling and material plasticity.

This approaches retains the simplicity of the modelling domain by restricting the deformation domain to axial loading (axial tension and compression), while including additional nonlinear regimes such as material plasticity and post-buckling, which cannot be modelled in traditional truss elements. The computational efficiency of the neural element can then be deployed into multi-truss structures (such as organic lattices) to overcome the high computational expense that is a current bottleneck in the modelling process.

To ensure an incremental design approach, the neural element will only be deployed on 2D (planar) multi-truss structures. If a positive outcome is achieved, then it may be deployed to 3D lattices and this will be addressed in this thesis. Additionally, it is assumed that all connection points between truss members (the red nodes on the far left of Figure 4.4) are pinned joints, and do not restrict rotational degrees of freedom.

4.2. Establishing the parameter design space

To proceed with the design of a neural element for multi-truss applications, the parameter design space has to be meticulously defined. As mentioned previously, the user-element has to model both tensile and compressive deformation. Throughout this section, the entire parameter design space will be specified. The design space refers to all the parameters flowing into and out of the neural network, and built within the respective user-subroutine; this is required to construct a machine learning framework for the neural network and the overall computational framework.

4.2.1. T2D2: The fundamental FEA truss element

In finite element modelling, the truss element is widely considered the simplest as it can only be loaded axially, meaning that it cannot deform out-of-plane like a beam element. In 2D-space, this can be mathematically translated into two degrees of freedom per node as shown in Figures 4.5 and 4.6.

Linear formulation

The two nodes of the truss element are denoted as 1 and 2 respectively, each of which is attributed two degrees of freedom in the local coordinate system. Therefore, the element as a whole possesses four degrees of freedom: $a_{e_{1x}}$, $a_{e_{1y}}$, $a_{e_{2x}}$, and $a_{e_{2y}}$. Intuitively, $a_{e_{1x}}$ and $a_{e_{2x}}$ are axially aligned with the truss which can induce axial tension and compression in the overall structure. The other components normal to the axis of the truss, $a_{e_{1y}}$ and $a_{e_{2y}}$ have no effect on the tensile deformation or compressive loads throughout the structure, but will rather cause rigid-body motion. It should be noted that up until this point, all notation concerns solely the truss' nodal displacements in the local coordinate system. In the global reference system, nodal displacements are applied as a_{1X}, a_{1Y}, a_{2X}, and a_{2Y} and need to be translated to the truss' local coordinate system depending on the angle θ between the reference frames as shown in Figure 4.5. Mathematically, this relation can be described by Equation 4.1.

$$\begin{aligned}
a_{1X} &= a_{e_{1x}} \cos\theta - a_{e_{1y}} \sin\theta \\
a_{1Y} &= a_{e_{1x}} \sin\theta + a_{e_{1y}} \cos\theta \\
a_{2X} &= a_{e_{2x}} \cos\theta - a_{e_{2y}} \sin\theta \\
a_{2Y} &= a_{e_{2x}} \sin\theta + a_{e_{2y}} \cos\theta
\end{aligned} \tag{4.1}$$

The nodal displacements can be translated into matrix form for efficient notation: $\mathbf{a} = \mathbf{T}_e\mathbf{a}_e$. Where $\mathbf{a}$ is the global displacement vector, $\mathbf{T}_e$ is the transformation matrix which rotates the global coordinate system to the local one, and $\mathbf{a}_e$ is the local displacement vector. Although $\mathbf{T}_e$ and $\mathbf{a}_e$ are element-specific, $\mathbf{a}$ assembles all the nodal degrees over all elements in a system. In the case of truss elements, if each node has two degrees of freedom and the system as a whole has N global degrees of freedom, then $\mathbf{a}_e$ is a 4-dimensional vector, $\mathbf{a}$ is an N-dimensional vector, and $\mathbf{T}_e$ is a $N \times 4$ matrix. These represent the nodal displacements, through the exact same approach can be used for the forces as well. The global force vector $\mathbf{f}$ is related to the nodal forces, $\mathbf{f}_e$ through the exact same transformation matrix, $\mathbf{T}$ such that: $\mathbf{f} = \mathbf{T}\mathbf{f}_e$.

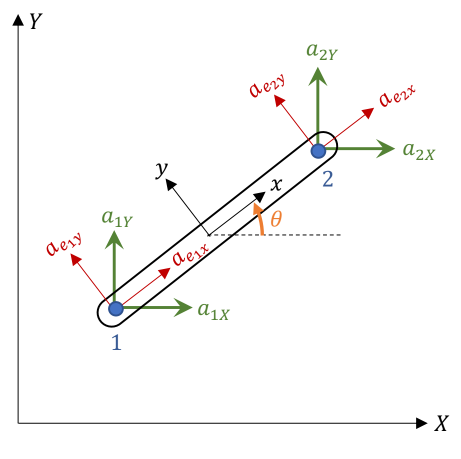


Figure 4.5: Truss element in 2D space, in local $(x - y)$ and global $(X - Y)$ reference frames (recreated from [107]).

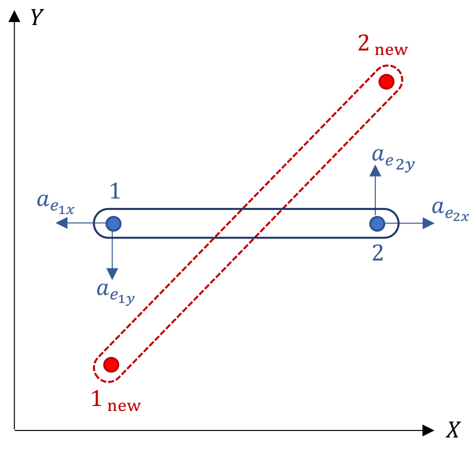


Figure 4.6: Example of a truss element subjected to combined loading causing an elongation and rotation.

It should be noted that this is a purely discrete notation of the element's displacement. Truss elements make use of simple linear interpolation between nodes expressed via shape functions h_k. For a single planar truss element, the continuous displacement field in local coordinates $\mathbf{u}_e$ can be expressed as (where η represents the 1D parametric coordinate system of a truss element):

$$\mathbf{u}_e = \sum_{k=1}^{2} h_k(\xi)\mathbf{a}_k \quad \rightarrow \quad \mathbf{u}_e = \mathbf{H}\mathbf{a}_e \tag{4.2}$$

In the context of linear modelling, the relationship between the forces and resultant displacement in local coordinates is described by $\mathbf{f}_e = \mathbf{K}_e\mathbf{a}_e$ where $\mathbf{K}_e$ is the element's stiffness matrix in local coordinates. For a T2D2 member, the stiffness matrix can be expressed through Equation 4.3 where E_e, A_e, and L_e denote the element's Young's modulus, cross-sectional area, and initial length respectively.

$$\mathbf{K}_e = \frac{E_e A_e}{L_e}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

In the global reference frame, the stiffness matrix of the element has to be altered using the transformation matrix: $\mathbf{f} = \mathbf{T}\mathbf{K}_e\mathbf{T}^\mathrm{T}\mathbf{a}$. If multiple elements are interconnected, then their individual stiffness matrices can be superimposed to create a global stiffness matrix $\mathbf{K}$. From there, the structural displacement $\mathbf{a}$ can be solved for any loading scheme $\mathbf{f}$. It is however important to note that this formulation is only valid in the linear deformation regime.

One final important aspect to underline is the possible deformation field of a truss member. Given its nodal degrees of freedom it is capable of axial shortening/elongation, and rigid body motion. Figure 4.6 provides a qualitative example of a truss element under combined loading. If the truss is loaded in such a way that the resultant displacement field is equal and opposite in both the x and y directions as displayed in the figure, then two events will occur simultaneously. The opposite axial loads $u_{e_{1x}}$ and $u_{e_{2x}}$ induce tension in the structure resulting in an elongation; whereas $u_{e_{1y}}$ and $u_{e_{2y}}$ cause a rigid-body rotation of the structure. The truss' final deformation state is shown in red in Figure 4.6.

Nonlinear formulation

The nonlinear finite element formulation differs slightly from the linear one, and has to be used when the relation $\mathbf{f} = \mathbf{Ka}$ no longer holds. This occurs in problems where material, geometrical, or contact nonlinearities occur. In such cases, the mathematical solver adopts an approach where the internal forces are balanced with the external forces: $\mathbf{f}_{\text{ext}} - \mathbf{f}_{\text{int}} = \mathbf{0}$. In a nonlinear model, a nonlinear continuum is discretized, thereby requiring an iterative approach to arrive at a solution. Moreover, since materials typically exhibit path-dependent behaviour, it is even more important to iterate the solver using relatively small strain values such that the final structural stress and strain paths are as close to reality as possible. The expressions for $\mathbf{f}_{\text{ext}}$ and $\mathbf{f}_{\text{int}}$ may be readily derived from the weak form of the finite element discretization in the initial configuration:

$$\int_{\Omega_0} \delta\boldsymbol{\varepsilon} \cdot \boldsymbol{\sigma} \mathrm{d}V_0 = \int_{\Omega_0} \delta\mathbf{u} \cdot \mathbf{b}_0 \mathrm{d}V_0 + \int_{\Gamma_0} \delta\mathbf{u} \cdot \mathbf{t}_0 \mathrm{d}A_0 \tag{4.4}$$

Where Ω_0 and Γ_0 represent the initial domain and boundary, and $\mathbf{b}_0$ and $\mathbf{t}_0$ are the initial body force densities and engineering stress. The external force vector $\mathbf{f}_{\text{ext}}$ may be expressed as follows:

$$\mathbf{f}_{\text{ext}} = \int_{\Omega_0} \mathbf{H}^T \mathbf{b}_0 \mathrm{d}V_0 + \int_{\Gamma_0} \mathbf{H}^T \mathbf{t}_0 \mathrm{d}A_0 \tag{4.5}$$

On the other hand, the internal force vector $\mathbf{f}_{\text{int}}$, is expressed as:

$$\mathbf{f}_{\text{int}} = \int_{\Omega_0} \mathbf{B}^T \boldsymbol{\sigma} \mathrm{d}V_0 \tag{4.6}$$

The stiffness matrix $\mathbf{K}$ which was introduced in the linear formulation is also included in the nonlinear formulation and consists of two components: the geometrical stiffness matrix ($\mathbf{K}^{\text{Geo}}$), and the material stiffness matrix ($\mathbf{K}^{\text{Mat}}$). These two components are added together to form the overall stiffness matrix: $\mathbf{K} = \mathbf{K}^{\text{Geo}} + \mathbf{K}^{\text{Mat}}$. The full expression is expressed as follows:

$$\mathbf{K} = \frac{\partial \mathbf{f}_{\text{int}}}{\partial \mathbf{a}} = \int_{\Omega_0} \left(\frac{\partial \mathbf{B}^T}{\partial \mathbf{a}}\right) \boldsymbol{\sigma} \mathrm{d}V_0 + \int_{\Omega_0} \mathbf{B}^T \left(\frac{\partial \boldsymbol{\sigma}}{\partial \mathbf{a}}\right) \mathrm{d}V_0 \tag{4.7}$$

From here, the iterative incremental solver using the Newton-Rhapson method can be initiated. For each loading increment, the following numerical scheme takes place (iterative analysis starts at $i = 0$:

1. Initialise data for loading step by setting $\Delta\mathbf{a}_0 = \mathbf{0}$
2. Compute the new external force vector $\mathbf{f}_{\text{ext}}^{t+\Delta t}$
3. Compute the initial residual force vector, $\mathbf{r}_i$ using the internal forces from the previous iteration (denoted as $\mathbf{f}_{\text{int,i}}$):

   $\mathbf{r}_0 = \mathbf{f}_{\text{ext}}^{t+\Delta t} - \mathbf{f}_{\text{int,0}}$

4. Determine the tangential stiffness $\mathbf{K}_i$:

   $$\mathbf{K}_i = \frac{\partial \mathbf{f}_{\text{int}}(\mathbf{a}^t + \Delta\mathbf{a}_i)}{\partial \mathbf{a}} = -\frac{\partial \mathbf{r}_i}{\partial \mathbf{a}}$$

5. Compute incremental displacement vector $\Delta\mathbf{a}_{i+1}$:

   $\Delta\mathbf{a}_{i+1} = \Delta\mathbf{a}_i + \mathbf{K}_i^{-1}\mathbf{r}_i$

6. Compute $\boldsymbol{\varepsilon}$, $\boldsymbol{\sigma}$, and $\mathbf{B}$

7. Compute internal force vector, summed over all integration elements and integration points:

$$\mathbf{f}_{\text{int},i+1} = \int_{\Omega_0} \mathbf{B}^T \boldsymbol{\sigma} \mathrm{d}V_0$$

8. Check convergence: $\| \mathbf{f}_{\text{ext}}^{t+\Delta t} - \mathbf{f}_{\text{int,i+1}} \| < \eta$ where η is a small number such as the convergence tolerance. If the convergence criteria are satisfied, then the iterative process is finished, and the solver proceeds to the next loading step. If not, then the iterative solver returns to Step 3.

This scheme is iterated until $\| \mathbf{f}_{\text{ext}}^{t+\Delta t} - \mathbf{f}_{\text{int,i+1}} \|$ tends to zero, or within an acceptable convergence margin, η. Another consideration is the stiffness matrix. It has to be factorized, which can become expensive for large systems. This will be further discussed when implementing the neural network. The Newton-Rhapson method is just one of many that is used in FEA nonlinear analyses. Others such as the Quasi Newton-Rhapson or Modified Newton-Rhapson methods improve the efficiency of the numerical scheme by altering the manner in which $\mathbf{K}_i$ is calculated. It is also important to understand how these linear and nonlinear schemes are built into an FEA solver which will be addressed shortly.

The type of element used in this example and brief introduction is designated as a T2D2 (2-node, linear 2D truss) element within ABAQUS' modelling environment, and it is the simplest element in finite element theory. There exist more complex variations of the truss element, such as T2D3 or T2D3H; however, those will not be required for the purposes of data generation as will be explained throughout this chapter. The importance of the T2D2 element in the context of this thesis is to emphasize the properties of a structural truss element in finite element formulation, as well as providing a benchmark case to which the capabilities of the neural network can be compared. Despite their simplicity, T2D2 elements represent the foundation of the finite element method and are widely used in FEA simulations to model and optimize multi-truss structures. Unlike the T2D2 element, the neural element will not make use of finite element formulation: from here on, a machine-learning perspective is required.

4.2.2. Standard user-element (UEL) formulation

Up until now, the neural element has been defined as a truss-like element that includes the deformation properties of beam elements. However, the key part of the neural element is that it leverages a neural network to bypass computational steps within the numerical solver during an FEA analysis. Therefore, to be able to define the parameter design space of the neural network, it is important to understand how such numerical solver schemes function at the user-element level. Consider the incremental-iterative Newton-Rhapson scheme in ABAQUS shown in Figure 4.7. The input file contains all the model details, such as the structure's geometry, material, mesh-discretization, number of elements and their respective locations, the boundary conditions, and other relevant user-defined parameters. It is also where the user can reference a user-subroutine to embed within the simulation. The user-subroutine cannot be directly written into the input file, but has to be created in a separate FORTRAN file. Once the model is loaded into ABAQUS' FEA environment via the input file, it is passed to the ABAQUS/Standard solver.

The solver starts by applying an initial external loading increment, followed by the initial residual force vector, and then initialises the data for the loading step. This is the exact process as outlined in previously in the numerical scheme for the Newton-Rhapson method. However, here a new notation is introduced to represent solely the element's displacements in the global reference frame:

$\tilde{\mathbf{a}}_e$. The use of a tilde accent serves to denote that a matrix/vector of reduced order (applied to a single element) is computed in the global reference frame. A brief summary of the used notation is shown below.

- $\mathbf{a}_e$: Nodal displacement vector of a single element in *local* coordinates.
- $\tilde{\mathbf{a}}_e$: Nodal displacement vector of a single element in *global* coordinates.
- $\mathbf{a}$: Nodal displacement vector of all nodal displacements in a model in *global* coordinates.

The user-element is then responsible for determining the stiffness matrix and internal force vector at the element level, $\tilde{\mathbf{K}}_{e,i}$ and $\tilde{\mathbf{f}}_{e,\text{int},i+1}$ respectively. Once completed, the ABAQUS solver assembles all the data into a global stiffness matrix and internal force vector ($\mathbf{K}_i$ and $\mathbf{f}_{\text{int},i+1}$), upon which the convergence test is performed. Depending on the outcome, the iterative process is continued to the following increment: $i = i + 1$, or the data is saved and the solver shifts to the following load increment. By understanding the UEL framework within ABAQUS, the requirements of the neural network used to build the neural element can now be established.

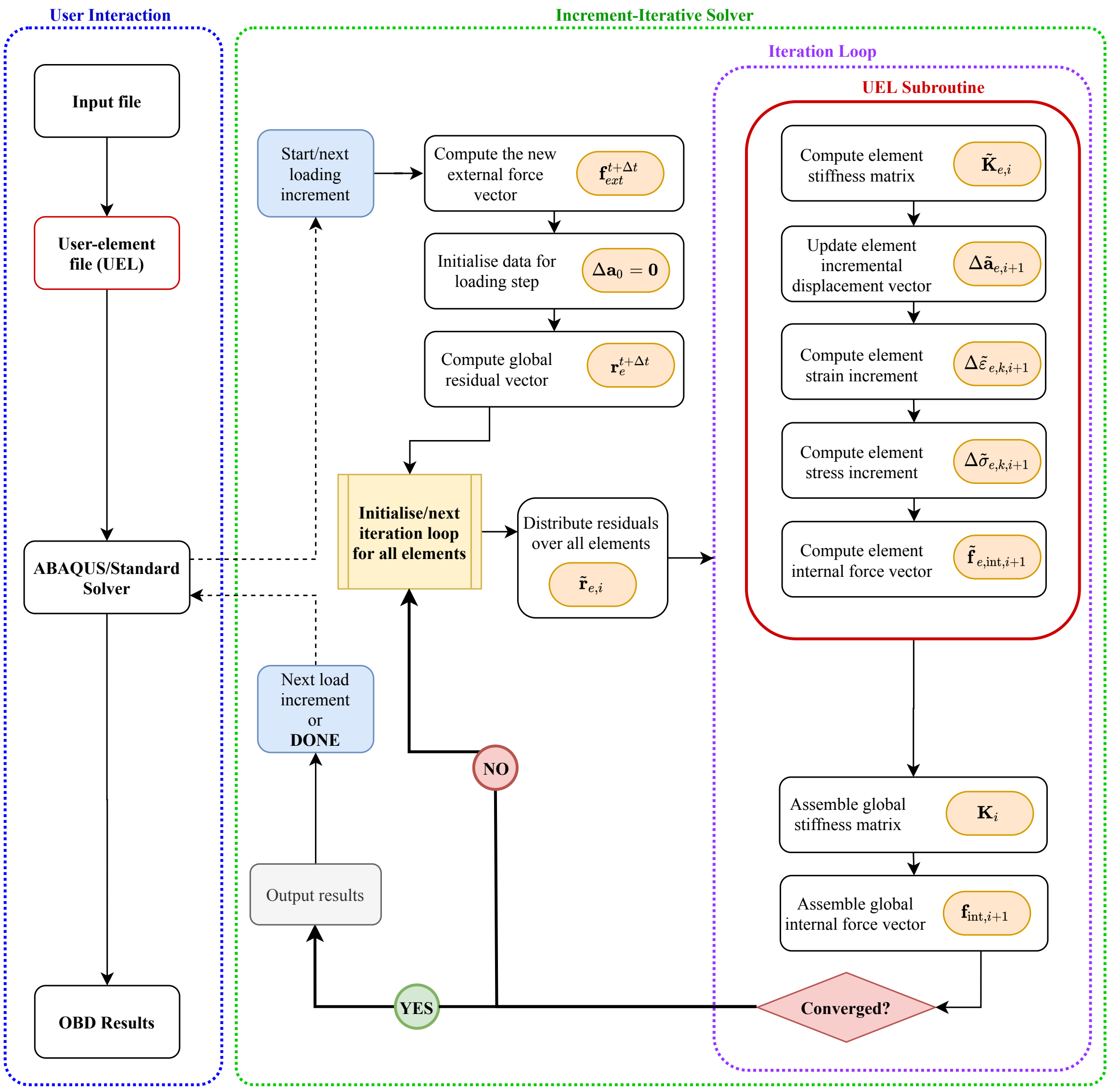

User Interaction
Increment-Iterative Solver
Iteration Loop
UEL Subroutine
Input file
User-element file (UEL)
ABAQUS/Standard Solver
OBD Results
Start/next loading increment
Compute the new external force vector $\mathbf{f}^{t+\Delta t}_{ext}$
Initialise data for loading step $\Delta\mathbf{a}_0 = \mathbf{0}$
Compute global residual vector $\mathbf{r}^{t+\Delta t}_e$
Initialise/next iteration loop for all elements
Distribute residuals over all elements $\tilde{\mathbf{r}}_{e,i}$
Compute element stiffness matrix $\tilde{\mathbf{K}}_{e,i}$
Update element incremental displacement vector $\Delta\tilde{\mathbf{a}}_{e,i+1}$
Compute element strain increment $\Delta\tilde{\varepsilon}_{e,k,i+1}$
Compute element stress increment $\Delta\tilde{\sigma}_{e,k,i+1}$
Compute element internal force vector $\tilde{\mathbf{f}}_{e,\text{int},i+1}$
Assemble global stiffness matrix $\mathbf{K}_i$
Assemble global internal force vector $\mathbf{f}_{\text{int},i+1}$
Converged?
YES
NO
Next load increment or DONE
Output results

Figure 4.7: Flowchart of the solver process within ABAQUS using an interfaced user-element (UEL) subroutine.

4.2.3. Neural Network user-element (UEL) formulation

As outlined in Chapter 2, neural networks rely on large amounts of training data to build a robust prediction tool. To design a neural element, the finite element approach has to be reformulated in terms of raw inputs and outputs into and out of the network. The iterative loop within the numerical solver is typically the source of high computational cost for an FEA simulation. This high cost could be due to the extensive iterations required within a loading step to achieve convergence, or rather due to the inversion process of the stiffness matrix to compute the residual vector, $\mathbf{r}_i$. Therefore, creating a neural network in the user-element has significant potential to reduce the computing time.

The two key entities that are required from a user-element are the element-wise stiffness matrix and internal force vector ($\tilde{\mathbf{K}}_{e,i}$ and $\tilde{\mathbf{f}}_{e,\text{int},i+1}$) to form the global stiffness matrix and internal force vector ($\mathbf{K}_i$ and $\mathbf{f}_{\text{int},i+1}$). The other steps in the numerical scheme are required to generate these outputs, but they do not necessarily need to be included as long as the outputs of the UEL comply with the ABAQUS solver to conduct the convergence test. One possible scenario is to have a neural network that is capable of directly computing the elemental stiffness matrix and internal force vector. The required inputs for this network would be the element's geometry, its material properties and its spatial positioning all of which influence its stiffness matrix. Additionally, the neural network would require the nodal displacements as input, which would allow the determination of the internal force vector.

Unfortunately, this solution is not feasible when considering the required training domain. Building a training dataset for such a neural network would require performing FEA analyses on truss members of various geometries, material properties and nodal displacement combinations. All of this is possible, however, the dimensionality of the training domain explodes when including spatial positioning. This phenomena was introduced in Chapter 2 and is commonly referred to as the *curse of dimensionality* [15]. As seen before, in both the linear and nonlinear formulations, the numerical solver requires the elemental stiffness matrix in global coordinates, which in turn requires the element's angular position relative to the global reference frame. Considering that all these inputs makes the training domain far too large, any gains in computational efficiency by the neural network in comparison to the traditional FEA model will be offset by the time required to build the training data.

A logical step is to scale down the required inputs and outputs of the network by removing the spatial positioning from the input data, although then it will not be possible to determine $\tilde{\mathbf{K}}_{e,i}$. The stiffness matrix of the truss member in local coordinates ($\mathbf{K}_{e,i}$) can still be determined, though it could be computed simply, rather than being passed through a network. Additionally, assuming that the entire structure is composed of the same isotropic material (metal for example), then the

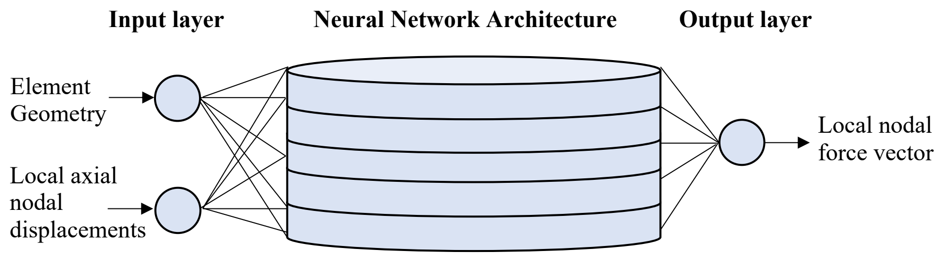


Figure 4.8: Example of a possible neural network within a UEL to compute the internal force vector as a function of the element geometry and nodal displacements.

material properties can be removed from the input layer of the neural network. The ability of the network to output the entire internal force vector in global coordinates is also stringent. In the case of multi-truss structures, a more practical output and simpler for the network to compute is the internal nodal force of one of the truss' nodes (node 1 for example) in *local* coordinates, denoted as $f^{\text{ANN}}_{e,\text{int},i+1_1}$. The final proposed input-output configuration is shown in Figure 4.8.

The proposed network in Figure 4.8 provides the nodal force as an output but the user-element is still required to output the elemental stiffness matrix and internal force vector in global coordinates. Since the neural network disregards spatial positioning, it is no longer possible to determine the tangential elemental stiffness matrix as used in the Newton-Rhapson method, which is required to build the internal force vector.

One option to circumvent this problem is the Quasi Newton-Rhapson method, which computes the secant stiffness as opposed to the tangential stiffness [32]. It might require more iterations until convergence, but ultimately achieves the same goal. Furthermore, it can be recursively computed from a singular nodal force in the truss' local coordinate system, $f^{\text{ANN}}_{e,\text{int},i+1_1}$. An extensive explanation of the Quasi Newton-Rhapson method and how it measures-up to the traditional Newton-Rhapson method may be found in [19]. The calculation of the elemental material stiffness matrix in local coordinates using this approach when compared to the linear computation may be described as follows:

$$\mathbf{K}^{\text{Mat}}_{e} = \frac{E_e A_e}{L_e}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{nonlinear}} \mathbf{K}^{\text{Mat}}_{e,i} = -\frac{f^{\text{ANN}}_{e,\text{int},i+1_1}}{\Delta a_{e,\text{axial},i+1}}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

Where $\Delta a_{e,\text{axial},i+1}$ is a scalar representing the truss' axial elongation/shortening and can be computed from the local nodal displacements such that: $\Delta a_{e,\text{axial},i+1} = \Delta a_{e,i+1_{2x}} - \Delta a_{e,i+1_{1x}}$. Recall that $\Delta a_{e,i+1_{1x}}$ and $a_{e,i+1_{2x}}$ represent the axial displacement components of the element's nodal displacement vector in local coordinates, $\Delta \mathbf{a}_{e,i}$. This reformulation of the local material stiffness matrix leverages the $f^{ANN}_{e,\text{int},i+1_1}$ output of the neural network by subsequently computing the truss' secant material stiffness matrix in its local coordinate system. In the nonlinear formulation, however, it is also required to compute the geometrical stiffness matrix. In the local coordinate system, this is expressed as:

$$\mathbf{K}^{\text{Geo}}_{e,i} = -\frac{f^{\text{ANN}}_{e,\text{int},i+1_1}}{L_e}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{4.9}$$

From here the element's stiffness matrix in global coordinates can be assembled as follows:

$$\tilde{\mathbf{K}}_{e_i} = \mathbf{T}_e(\mathbf{K}^{\text{Mat}}_{e_i} + \mathbf{K}^{\text{Geo}}_{e_i})\mathbf{T}^{\text{T}}_e \tag{4.10}$$

Using this approach, the neural element will have the correct outputs to provide to the ABAQUS solver. The individual processes for the neural element have been successfully identified and constitute the neural element.

The entire process and framework for the neural element are summarized in Figure 4.9. The custom element leverages the expressive power of a neural network to compute quickly the nodal forces which are then used to build the required outputs for the ABAQUS solver. One noteworthy feature of the UEL is that in the case of the initial iteration ($\Delta\mathbf{a}_0 = \mathbf{0}$), the stiffness matrix is computed using the linear formulation to prevent singularities in the numerical solver which would be obtained when using the Quasi Newton-Rhapson approach. Although the new framework appears more complex than the traditional UEL, the neural network allows for higher computing speeds in addition to the fact that the neural element models an entire truss member rather than just a single element within the truss.

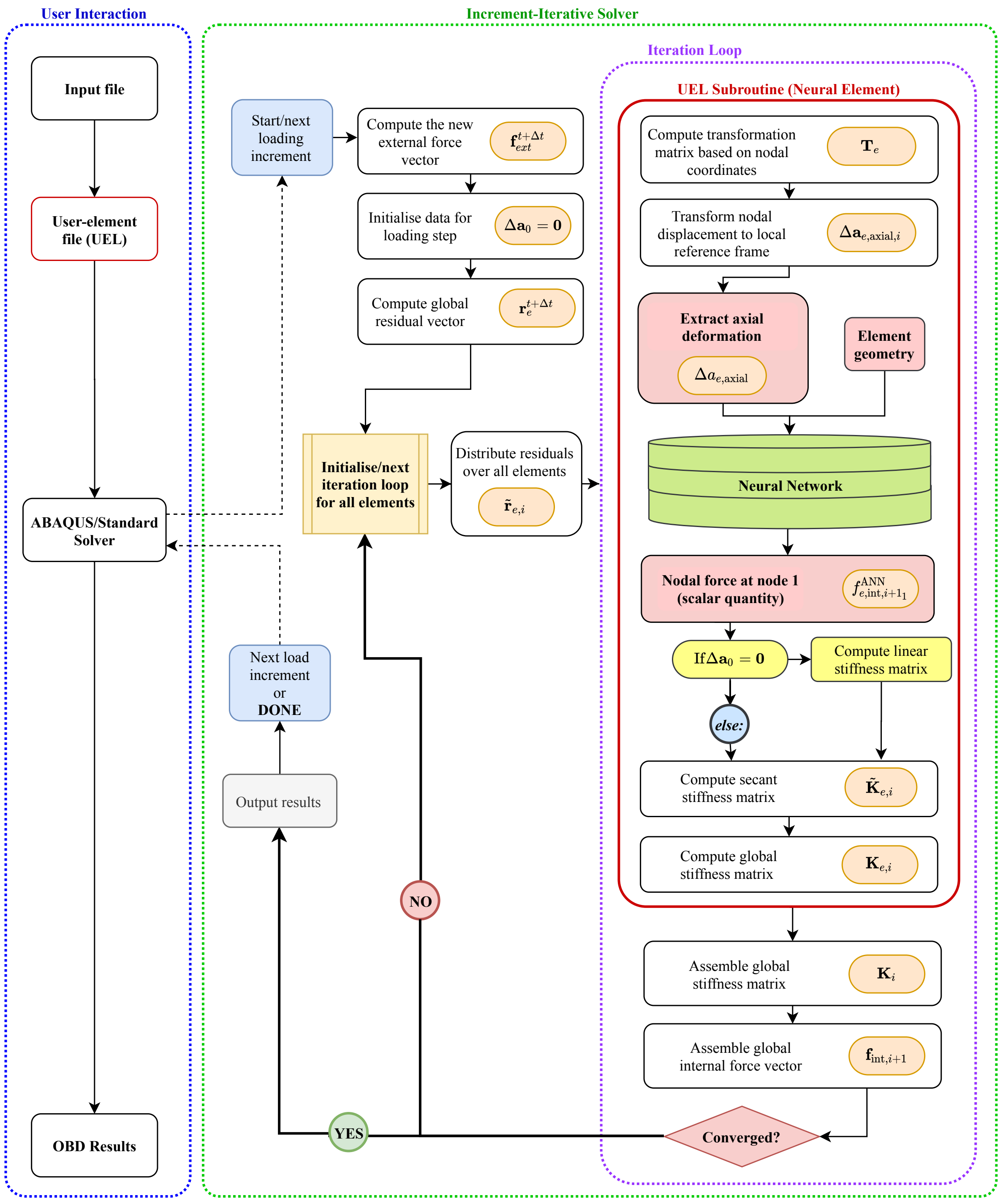

User Interaction
Increment-Iterative Solver
Iteration Loop
UEL Subroutine (Neural Element)
Input file
User-element file (UEL)
ABAQUS/Standard Solver
OBD Results
Start/next loading increment
Compute the new external force vector
$\mathbf{f}^{t+\Delta t}_{ext}$
Initialise data for loading step
$\Delta\mathbf{a}_0 = \mathbf{0}$
Compute global residual vector
$\mathbf{r}^{t+\Delta t}_{e}$
Initialise/next iteration loop for all elements
Distribute residuals over all elements
$\tilde{\mathbf{r}}_{e,i}$
Next load increment or DONE
Output results
NO
YES
Compute transformation matrix based on nodal coordinates
$\mathbf{T}_e$
Transform nodal displacement to local reference frame
$\Delta\mathbf{a}_{e,\text{axial},i}$
Extract axial deformation
$\Delta a_{e,\text{axial}}$
Element geometry
Neural Network
Nodal force at node 1 (scalar quantity)
$f^{\text{ANN}}_{e,\text{int},i+1_1}$
If $\Delta\mathbf{a}_0 = \mathbf{0}$
Compute linear stiffness matrix
else:
Compute secant stiffness matrix
$\tilde{\mathbf{K}}_{e,i}$
Compute global stiffness matrix
$\mathbf{K}_{e,i}$
Assemble global stiffness matrix
$\mathbf{K}_i$
Assemble global internal force vector
$\mathbf{f}_{\text{int},i+1}$
Converged?

Figure 4.9: Flowchart of the solver process within ABAQUS using the interfaced neural element.

4.2.4. Geometrical boundaries

The final stage in defining the parameter design space of the neural element concerns the geometrical boundaries of the truss member onto which the element is deployed. The neural network can only be applied to truss members that are within the boundaries of its training domain. Hence, the geometrical boundaries have to be prescribed, because the network cannot be expected to model any random configuration (especially at this early development stage). The geometrical boundaries of truss members are globally defined by the suggested context and their associated manufacturing process: biomimicry and 3D-printing.

3D-printing imposes a material constraint which is that the entire structure is made from the same isotropic material: typically either metal or plastic. Although plastics are one of the most common materials for 3D-printing, they are disregarded for this thesis. Small slender, truss members made from polymeric plastics have the potential to exhibit hyper-elastic tendencies that require dedicated material models. This complexity can be avoided by modelling the structures out of metal (Figures 4.10 and 4.11 show an example of powder bed fusion which is one method used to 3D-print metal structures). Steel is the chosen material to build the parameter design space of the neural network since it represents an available material in 3D-printing [121]. The common use of steel in manufacturing coupled with its known specific properties implies that it can be used for a variety of applications including multi-truss organic structures as well as other demanding industrial uses.

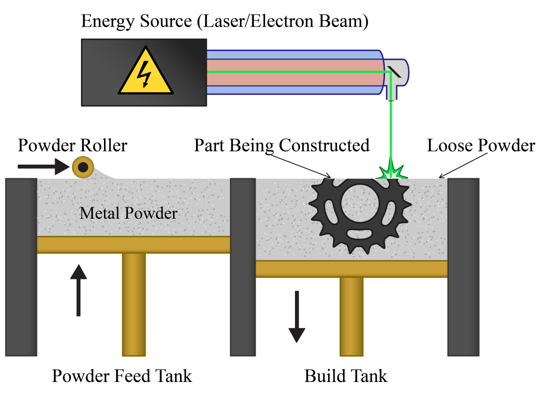


Figure 4.10: Example of a metal 3D-printing process using powder bed fusion [2].

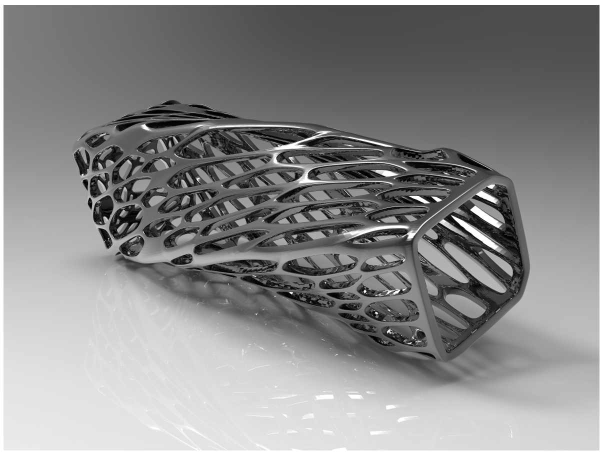


Figure 4.11: Possible product resulting from a metal 3D-printed process (rendered in KeyShot).

The 3D-printing manufacturing context also introduces another variable: structural defects manifested through material porosity. Porosity defects are regions within the final product of lower material density. These can lower the structure's mechanical properties and are difficult to detect. Despite current research and technological advancements, such defects are still a problem in 3D-printed components [2]. Although it is difficult to model porosity, one possible simplification is to have a defect radius, $r_{e,D}$, in the middle of the truss member that is smaller than the overall elemental radius: $r_{e,D} < r_e$. This creates a weak link in the truss member thereby reducing its mechanical properties. The relationship between porosity and defect radius in a truss member is not yet established, the inclusion of a defect radius provides the end-user with the potentially useful option to introduce such a defect in the structure.

Given the above considerations, the final geometrical design space of the truss member includes the length and nominal radius, denoted as L_e and r_e respectively, as well as a defect radius: $r_{e,D}$. The chosen boundaries are such that a multi-truss structure remains relatively small. Figure 4.12 provides a diagram of a single truss element and its respective variables. Table 4.1 summarizes all the variables and their defined boundaries. The overall length L_e is capped between 10.0 and 30.0mm, and the radius r_e is bounded by 0.5 and 2.0mm. The defect radius $r_{e,D}$ is defined as a function of r_e such that the largest radial defect is equal to 50% of r_e. An undamaged truss member possesses no radial defect, therefore in such a case: $r_{e,D} = r_e$. These three parameters are raw inputs into the neural network, and when combined with the axial displacement, the nodal force at node 1 will be the output of the network. An additional constraint must be defined: the portion of the truss that is subjected to $r_{e,D}$. The length, $L_{e,D}$ is constrained to 5% of the overall length, i.e., $L_{e,D} = 0.05L_e$. The reason $L_{e,D}$ is not a variable in the element's geometry is to maintain a relatively small design space. Since it is an imposed constraint, it is greyed-out in both Figure 4.12 and Table 4.1.

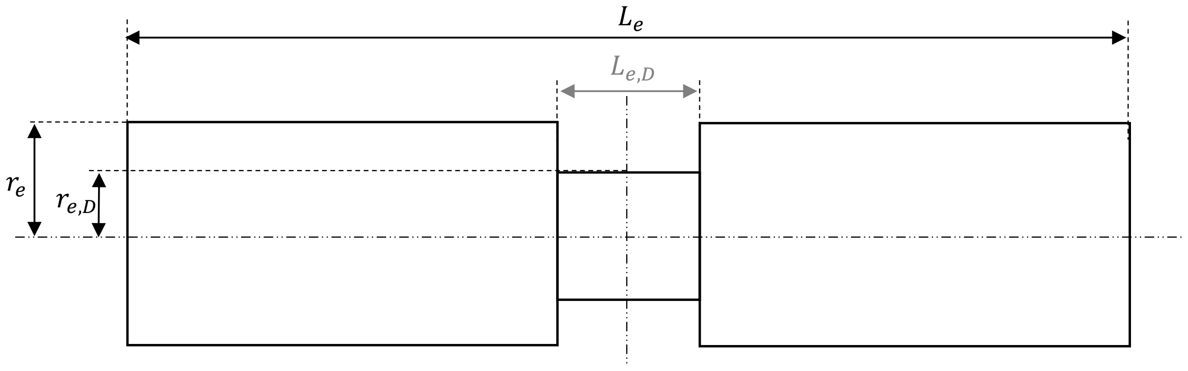


Figure 4.12: Overview of a truss element's geometrical variables (imposed constraints are greyed-out and cannot be altered by the end-user).

The final boundary introduced at this stage, because it is directly related to the geometrical boundaries is the axial displacement $\Delta a_{e,\text{axial}}$. Note that this is slightly different from the previous notation: $\Delta a_{e,\text{axial},i+1}$. The $i+1$ subscript has been removed for clarity since it refers to the iteration stage in the numerical solver scheme. For the remainder of this thesis, the axial elongation/contraction caused by applied loads, will be referred to as $\Delta a_{e,\text{axial}}$ and is present in Table 4.1 (note that it is also greyed-out since it is determined by the global geometry of the truss member). This parameter governs the resultant nodal force and will influence the deformation field of the neural element. The boundary should be such that it is dependent on the geometrical parameters of the truss, and is sufficient to induce nonlinear behaviour and not remain in the linear-elastic regime. Therefore the chosen boundary is a function of the defect length of the structure and is set to $\Delta a_{e,\text{axial}} \in \pm[0, 0.5L_{e,D}]$. The reason for the $\pm$ sign is that the structure can be loaded in either tension or compression.

Table 4.1: Neural network inputs and their respective boundaries (imposed constraints are greyed-out and cannot be altered by the end-user).

Input Variable	**Symbol**	**domain**
Length	L_e	$L_e \in [10.0\text{mm}, 30.0\text{mm}]$
Radius	r_e	$r_e \in [0.5\text{mm}, 2.0\text{mm}]$
Defect Radius	$r_{e,D}$	$r_{e,D} \in [0.5r_e, r_e]$
Defect Length	$L_{e,D}$	$L_{e,D} = 0.05L_e$
Axial Displacement	$\Delta a_{e,\text{axial}}$	$\Delta a_{e,\text{axial}} \in \pm[0, 0.5L_{e,D}]$

4.3. Building the global framework

Establishing the parameter design space is arguably one of the most important steps in the entire project. The end-goal is to produce a user-element that utilises a neural network to improve computational efficiency, which can only be achieved when the inputs/outputs of the neural network along with their boundaries are clearly formulated. If not properly done, then the design space of the neural networks risks becoming too big and the time required to generate the adequate training data will surpass any computational gains by the network [5].

However, despite its importance, the previous section is but a single step in the overall process. This section incorporates the neural element into a global computational framework. This framework outlines the necessary steps to build, deploy, and use the neural element in practical applications of multi-truss structures.

4.3.1. System settings

The global computational framework, in addition to addressing the separate processes that involve building and deploying the neural element, must also include the system settings from a hardware and software point of view to ensure replicability of the results. This is especially important since this is a purely computational topic and every system setting can influence the computational cost of an FEA simulation, or the speed at which a neural network is built.

Software

The principle FEA software used throughout this project is ABAQUS by Dassault Systèmes. All FEA simulations are run locally without requiring cluster-based access. It should be noted to build subroutines in ABAQUS (UEL, UMAT, etc.), cluster-based access is required. However, to avoid queue times and have access to ABAQUS' GUI after a simulation with a custom subroutine, the need for cluster-based access was circumvented by internally linking ABAQUS with a FORTRAN and C++ compiler.

From a coding perspective, Python provides some powerful modules to build and optimize a neural network such as *Keras*, *TensorFlow* and *Talos*. However, these are not suited to writing a user-element that requires a dedicated FORTRAN compiler. In such a case, Microsoft Visual Studio 2013 along with Intel Parallel Studio XE 2016 proved to be an adequate choice. All the used software is summarized in Table 4.2.

Table 4.2: Software used throughout the thesis.

Task	Used Software
FEA	ABAQUS
Neural Network development and optimisation	Python 3 with *Keras*, *TensorFlow*, and *Talos*
UEL development	ABAQUS, Microsoft Visual Studio 2013, and Parallel Studio XE 2013

Table 4.3: Hardware used throughout the project.

Device	Apple MacBook Pro 15" (late 2018)
Processor	2.9 GHz Intel Core i9
Memory	32 GB 2400 MHz DDR4
CPU	12 Cores (with Intel-based hyper-threading)
GPU	Radeon Pro Vega 20 4 GB Intel UHD Graphics 630 1536 MB

Hardware

All FEA analyses, neural network development and UEL development was conducted locally. The relevant hardware specifications are summarized in Table 4.3. All FEA analyses are conducted on a Windows 10 partition interfaced via Parallel Desktop 14.

4.3.2. Global computational framework

The global computational framework can be divided into four different steps: data generation, training the neural network, deploying in neural network inside the neural element, and finally using the neural element in an FEA setting. Figure 4.13 describes the various steps and how they interact with each other, all of which are described in greater detail below.

Step 1: Data Generation

To build and train a neural network for a given application, there has to be enough training data pertinent to such an application to ensure that the final deployed network is sufficiently robust. First a combination of the element's geometrical parameters is selected within their defined boundaries and a corresponding FEA model is built within ABAQUS. The structure representing a single truss member of length L_e, radius r_e and defect radius $r_{e,D}$ is then subjected to two isolated analyses: one in tension, and the second in compression. Once the FEA simulations complete, the force-displacement curve is extracted which represents the axial deformation of the truss member $\mathbf{a}_{e,\text{axial}}$ versus the internal force at node 1, $\mathbf{f}_{e,\text{int}_1}$. It is important to note that these two entities are arrays of values over the course of the analysis, i.e.: $\mathbf{a}_{e,\text{axial}} = a^{t0}_{e,\text{axial}} + a^{t1}_{e,\text{axial}} + a^{t2}_{e,\text{axial}} + ...$, and similarly: $\mathbf{f}_{e,\text{int}_1} = f^{t0}_{e,\text{int}_1} + f^{t1}_{e,\text{int}_1} + f^{t2}_{e,\text{int}_1} + ...$ Therefore, after combining the data from both FEA analyses, there is a displacement field combining tension and compression, which corresponds to nodal forces at node 1 which include both plastic deformation under tension and the post-buckling regime, for a given truss member specific geometric parameters L_e, r_e, and $r_{e,D}$.

This process is repeated for all geometrical configurations within the total parameter design space. Then the quality of the total data repository is evaluated in terms of suitability for neural network training. This is clarified in Chapter 5, however a brief example is that the post-buckling regime is typically more nonlinear than the plastic deformation in tension for an isotropic beam-like structure. Therefore, the numerical solver is required to take more loading steps to capture the nonlinear regions in compression than in tension. The resultant dataset for an element with selected geometrical parameters will have far more data-points corresponding to compression than tension. If the data are randomly sampled, then this introduced an unintentional bias that in the global dataset which would make the subsequent neural network favour compression over tension. Hence assessing the quality of the data for a supervised learning process is a crucial step. If the data are considered inadequate, then the FEA models can be altered to correct for this, or, it is also possible to alter the sampling process of the neural network at a later stage. This is discussed in greater detail in Chapter 6 which focuses on building and training the neural engine.

Step 2: Building the neural network

Once the data depository has been built, it can be passed onto the neural network which builds and trains a neural network. The first stage is reformatting the data in terms of inputs and outputs. For truss members with a geometrical and displacement at time t, there is a nodal force at node 1. The inputs/outputs can be summarized as follows:

- **Neural network input**: $\left[L_e \;\; r_e \;\; r_{e,D} \;\; \Delta a^t_{e,\text{axial}}\right]^{\mathrm{T}}$
- **Neural network output**: $\left[f^{\text{ANN},t}_{e,\text{int}_1}\right]$

Once the inputs/outputs are formatted correctly, the data is then pooled, normalized, and randomly split into training, validation and testing datasets. The ratios and techniques used to properly split the global dataset were outlined in Chapter 2 and are reiterated in Chapter 6. The training and validation datasets are passed to the neural network and to a multi-objective optimization scheme. The latter aims to interpret the data and design architecture of the network so that the user does not have to pick every hyperparameter through a conventional trial-and-error approach. Once a suitable network is built and trained, its performance is assessed with the testing data. If the performance and expressivity of the network are unacceptable, then the boundaries of the multi-objective optimization scheme will be slightly altered. However, given the effectiveness of multi-objective optimization, it will be unlikely that the neural network has to be revised. Once an acceptable performance is achieved, then the neural network is deployed into the next stage of the overall process.

Step 3: Deploying the neural element

Due to its intricacy, the framework of the neural element was attributed a dedicated section and outlined in Figure 4.9. This step involves writing the custom user subroutine in FORTRAN and embedding the trained neural network in the user-element. Hereafter it can be deployed into the ABAQUS/ Standard solver as a user-element.

Step 4: Applying the neural element to multi-truss structures

The final step in the overall framework is applying the neural element to a variety of test cases and comparing the computational performance against that of an FEA simulation with traditional elements. Once a multi-truss structure is selected, two FEA models are built in ABAQUS. One is meshed with traditional elements, whereas the other is meshed with the neural element. The same boundary conditions are applied to both models and the simulations are initiated. Once completed, the models are compared both in terms of computational efficiency, and accuracy. The exact metrics that are evaluated are discussed for each case study respectively in order to determine which metric is/are most representative.

These four steps constitute the entire computational architecture that is used in this thesis. Figure 4.13 provides a global overview to illustrate how one process interacts with and flows into another.

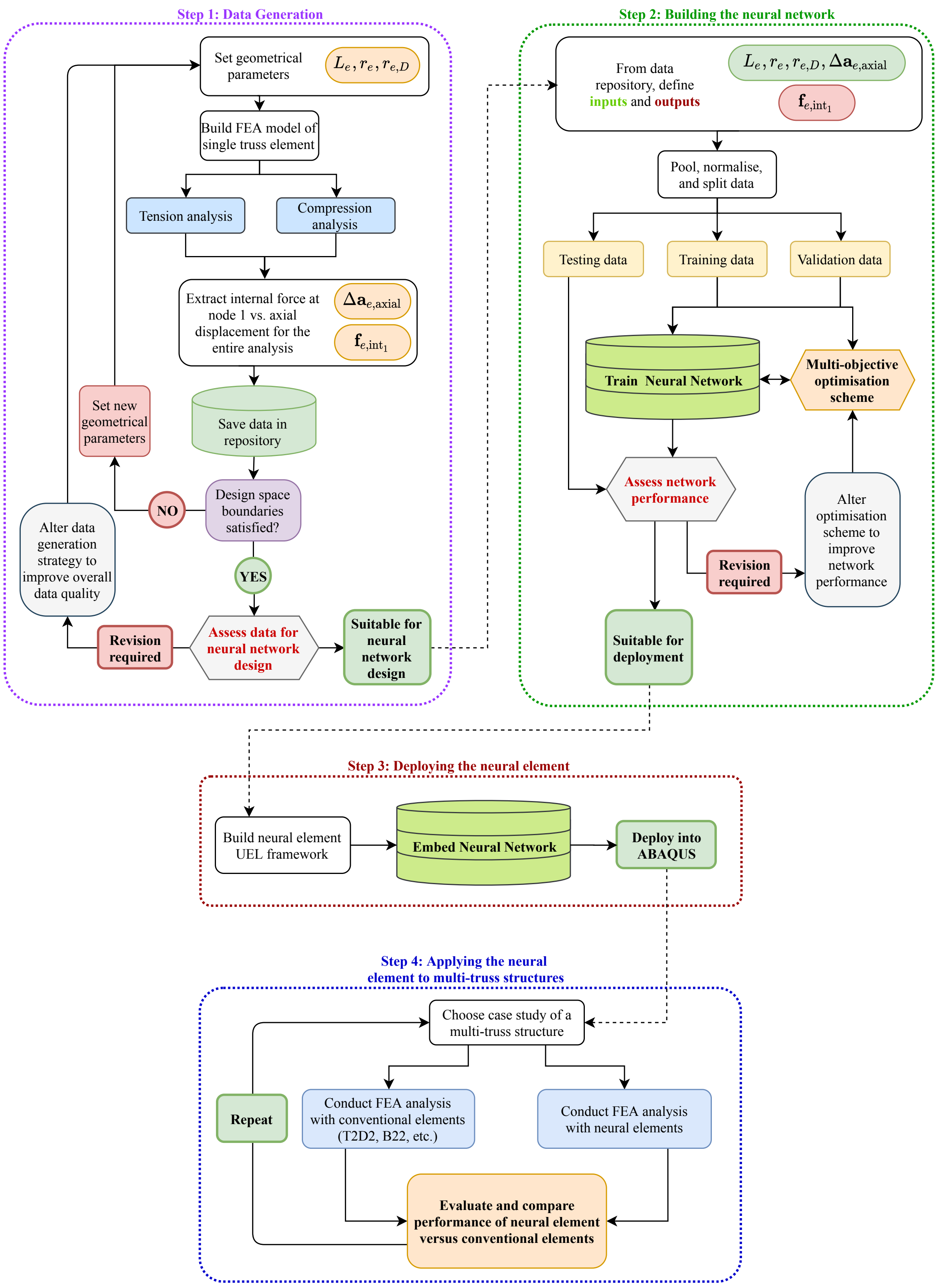

Step 1: Data Generation
Set geometrical parameters
$L_e, r_e, r_{e,D}$
Build FEA model of single truss element
Tension analysis
Compression analysis
Extract internal force at node 1 vs. axial displacement for the entire analysis
$\Delta\mathbf{a}_{e,\text{axial}}$
$\mathbf{f}_{e,\text{int}_1}$
Save data in repository
Design space boundaries satisfied?
NO
YES
Set new geometrical parameters
Alter data generation strategy to improve overall data quality
Revision required
Assess data for neural network design
Suitable for neural network design
Step 2: Building the neural network
From data repository, define inputs and outputs
$L_e, r_e, r_{e,D}, \Delta\mathbf{a}_{e,\text{axial}}$
$\mathbf{f}_{e,\text{int}_1}$
Pool, normalise, and split data
Testing data
Training data
Validation data
Train Neural Network
Multi-objective optimisation scheme
Assess network performance
Revision required
Alter optimisation scheme to improve network performance
Suitable for deployment
Step 3: Deploying the neural element
Build neural element UEL framework
Embed Neural Network
Deploy into ABAQUS
Step 4: Applying the neural element to multi-truss structures
Choose case study of a multi-truss structure
Repeat
Conduct FEA analysis with conventional elements (T2D2, B22, etc.)
Conduct FEA analysis with neural elements
Evaluate and compare performance of neural element versus conventional elements

Figure 4.13: Global computational framework to build, deploy and test the neural element on practical case studies.

4.4. Closure

The required process to build the neural element is complex and involves multiple processes across different platforms. The global computational architecture described in the previous section and outlined in Figure 4.13 provides an overview of the reach of this project. In summary, the neural element aims to achieve the following:

1. Circumvent the need for complicated mesh discretization schemes that are usually required to model nonlinear deformation in multi-truss structures. Instead, each truss member may be directly replaced by a single neural element.
2. The neural element will also permit the user to include material defects in the analysis since the trained neural network will have been designed to do so. Building FEA models with traditional finite element that take into account material defects introduces another layer of complexity that consumes more time in the overall process. Having a neural element with built-in capability to model defects represents a significant amount of time saved just in the model development phase.
3. The neural element will also take into account nonlinear deformation including the post-buckling regime and material plasticity.
4. Reduce the overall computational time of FEA simulations by using a neural network to bypass the computation of local nodal forces and the stiffness matrix.

It is also important to reiterate that throughout the remainder of this thesis, a machine-learning perspective should be adopted. Neural networks are capable of modelling the most difficult regression problems, though they are purely data-driven systems. If the training data is inadequate, unbalanced, or displays inherent bias, then a trained network will not perform well. In other words: "*garbage in, garbage out*".

5 Data Generation & Automation

Prior to commencing the training procedure of the neural engine, it is crucial to generate a sufficient amount of useful data. The trained network will then be deployed as a user-element within ABAQUS' FEA environment so as to improve the computational efficiency of multi-truss structures. Although outlined in the previous chapter, it bears repeating that the custom element will possess modelling capabilities outside the deformation regime of a typical truss element. Notably, it can treat the out-of-plane displacement during buckling and post-buckling when compressed, which is found in beam elements. The *truss* designation refers to an imposed restriction that the user-element can only take-up loads axially. Therefore, the resultant modelling domain of the network has to capture the linear and nonlinear deformation regimes of a beam under axial tension and compression. To reiterate from the previous chapter, the full deformation field of the neural truss element encompasses:

- Axial tensile deformation including material plasticity.
- Axial compressive deformation including post-buckling and material plasticity.

Capturing both tensile and compressive deformation becomes a complex regression problem for a neural network, especially given the degree of nonlinearity in each case. Not only will the user-element have to predict the plastic deformation when stretched, but also be able to model the post-buckling deformation when compressed. Both cases represent nonlinear behaviours. This behaviour can vary drastically even with slight alterations in the model's geometry. Impacts can be more pronounced when considering the addition of a defect radius $r_{e,D}$, as will be seen later in this chapter. These requirements alone push the learning capabilities of a neural network to the edge.

Building a high-fidelity neural network capable of modelling such deformation regimes and match the accuracy of traditional FEM analyses requires a large data volume, which is created through thousands of FEM simulations using a bespoke automation framework. This chapter builds on the developed computational architecture (Chapter 4), by focusing on the data generation strategy that is used to build the extensive data repository for the neural network.

5.1. Boundary conditions

First, the notion of *useful data* has to be properly defined. In the context of supervised machine learning, useful data is defined as: *any data sample within the predefined parameter design space that can single-handedly assist in improving the overall accuracy and expressivity of a neural network*

[5]. It should be emphasized that accuracy and expressivity measure very different attributes of the network. Whereas accuracy is measured relative to the training dataset for a singular data sample, expressivity is the network's ability to predict robustly the overall trend for a regression problem (as seen in Chapter 2). The parameter design space was established in the previous chapter, though as a reminder, the inputs for a given input vector containing the truss member's geometrical parameters and axial deformation, the neural network is required to predict the internal nodal force:

- **Neural network input**: $\begin{bmatrix} L_e & r_e & r_{e,D} & \Delta a^t_{e,\text{axial}} \end{bmatrix}^\text{T}$
- **Neural network output**: $\begin{bmatrix} f^{\text{ANN},t}_{e,\text{int}_1} \end{bmatrix}$

With this in mind, the current section aims to provide an overview of the boundary conditions that are established throughout the data generation process. These can be grouped into two areas: structural boundary conditions, and the boundaries imposed on the global data density. It will also become clear why the neural network outputs the internal nodal force $f^{\text{ANN}}_{e,\text{int}_1}$ at node 1.

5.1.1. Structural boundary conditions

Whether in tension or compression, the boundary conditions remain the same throughout the data generation process. Consider a singular truss structure as shown in Figure 5.1, which could also be considered as a beam for the purposes of boundary condition definition. The beam is pinned at its leftmost node (node 1), and simply supported at its rightmost end (node 2). The structure is then subjected to an external displacement, $\Delta a_{e,\text{axial}}$, which could either load the structure under tension if positive or compression if negative (relative to the global coordinate system). Naturally, there will be a reaction force at node 1 to counteract the disturbance and equilibrate the structure. This reaction force corresponds to the internal nodal force that is required to train the network, f_{e,int_1}.

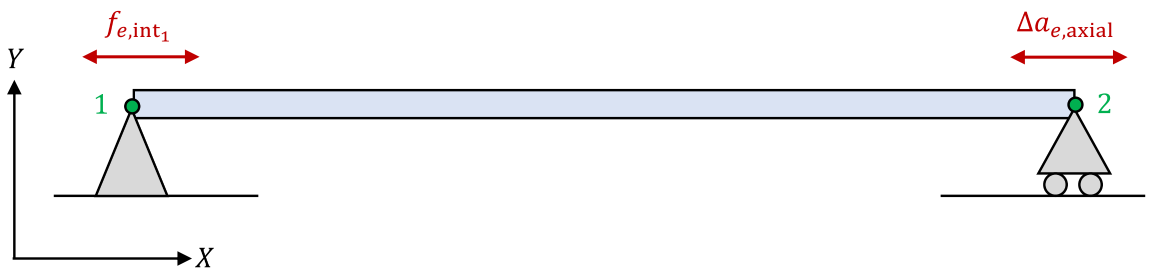


Figure 5.1: Structural boundary conditions imposed throughout the data generation process.

This simple set of boundary conditions is all that is required to commence the data generation phase. Depending on the direction $\Delta a_{e,\text{axial}}$, the approach will be tailored to suit a compressive or tensile deformation. It is also worth reiterating that the output of the trained neural network will consequently be $f_{e,\text{int}_1} \rightarrow f^{\text{ANN}}_{e,\text{int}_1}$ given a selected geometry and imposed external displacement of the structure, $\Delta a_{e,\text{axial}}$.

5.1.2. Capping the global data density

The global data density refers to the amount of data that will be generated throughout the entire data generation process. In the previous chapter, the boundaries of the parameter design space were established, however this did not include any boundaries on the amount of data to be generated. Since this is a purely numerical process, the amount of data to be generated is theoretically limitless, however, this would be a highly time-consuming process which defeats the purpose of the end-product: to be computationally efficient. Therefore, boundaries on the global data density have to be established before initiating the data generation process. The input variables to the neural network and their respective boundaries were defined in Chapter 4, in Table 4.1.

The neural network inputs create a 4-dimensional design space from which an infinite number of permutations can be drawn. Each permutation corresponds to a dataset that can be used to train the neural network through an additional iteration. Usually, the amount of required datasets to train a neural network adequately is closely related to the number of trainable parameters in the network as well as the number of independent input variables (which is 4 in our case). However, such an estimation is non-trivial given that the network architecture (and thereby the number of trainable parameters) will be determined through a multi-objective optimization process. Moreover, the output of the neural network will be capturing the internal nodal force throughout the deformation of a beam in either tension or compression, which are two highly nonlinear phenomena. Therefore, conventional estimates will not suffice to build the boundaries for the global data density.

Given the amount of expressiveness required of the neural network, a dense dataset is preferred over a sparse dataset. Therefore, the imposed boundaries on the geometrical parameters of a truss member are summarized below, in Table 5.1. The length, L_e will be iterated every 1mm. The nominal radius, r_e, every 0.1mm; and the defect radius, every 10%. This builds a simulation grid with 2,016 possible geometrical combinations. There is however, an additional input parameter that has to be considered: $\Delta a_{e,\text{axial}}$. Unfortunately, it is impossible to constrain the amount of data generated through $\Delta a_{e,\text{axial}}$ since it largely depends on the ABAQUS solver. Regions of high nonlinearity require a high sampling rate manifested by a small step size. If the minimum step size is not small enough, then the FEA simulation will likely abort stating problems of non-convergence. Though if the maximum step size is too small, then the model will likely take a very long time to complete its analysis. Furthermore, if the overall boundary for the step size is sufficiently large to prevent the previous two problems, then the resultant dataset will naturally be biased towards nonlinear regions, which pull the focus away from linear regions that are just as important. In summary, determining the optimal iteration boundary for $\Delta a_{e,\text{axial}}$ is a delicate balance of the following aspects:

- Ensuring convergence of the FEA model.
- Maintaining the computing time as low as possible.
- Preventing inherent bias in the overall dataset.

These will be revised in the following sections during convergence studies as well as the data repository analysis. For now, the iteration boundary will simply be labeled as *Case specific* as they

Table 5.1: Neural network input variables and their respective boundaries including iteration boundaries.

Input Variable	**Symbol**	**Domain**	**Iteration Boundary**
Length	L_e	$L_e \in [10.0\text{mm}, 30.0\text{mm}]$	Every 1mm $L = [10.0, 11.0, ..., 29.0, 30.0]$
Radius	r_e	$r_e \in [0.5\text{mm}, 2.0\text{mm}]$	Every 0.1mm $r_e = [0.5, 0.6, ..., 1.9, 2.0]$
Defect Radius	$r_{e,D}$	$r_{e,D} \in [0.5r_e, r_e]$	Every 10% $r_{e,D} = [0.5r_e, 0.6r_e, ..., 0.9r_e, r_e]$
Axial Deformation	$\Delta a_{e,\text{axial}}$	$\Delta a_{e,\text{axial}} \in \pm[0, 0.5L_{e,D}]$	*Case specific*

will depend on the geometrical parameters of each truss member. Though considering that for each geometrical combination of parameters L_e, r_e, $r_{e,D}$, both tension and compression will be considered, the projected number of datasets will be $4{,}032 \times \kappa$ where κ will be the number of iterations of $\Delta a_{e,\text{axial}}$ for each truss members. Given the degree of non linearity of this project, κ is expected to be anywhere between 30 and 200 depending on the convergence criteria of the developed models resulting in a global simulation grid size of at least 100,000 datasets. With the boundaries for the global data density established, the two loading cases can be discussed.

5.2. Axial tension: an overview

The first deformation mode considered is axial tension. From the boundary conditions established in Figure 5.1, a tensile loading occurs when $\Delta a_{e,\text{axial}} > 0$. As $\Delta a_{e,\text{axial}} > 0$ is increased, the internal reaction force at node 1, f_{e,int_1}, increases in the opposite direction to oppose the structural elongation. The axial displacement also induces a cross-sectional contraction that corresponds to the Poisson effect, which can be computed through the Poisson ratio: ν. Figure 5.2 shows a structure subjected to a positive axial placement causing a structural elongation. Note that the elongated section depicted in yellow is of smaller cross-sectional area, representing the Poisson effect. The imposed displacement also results in a new overall length $L_{e,\text{new}}$ such that $L_{e,\text{new}} = L_e + \Delta a_{e,\text{axial}}$. For small values of $\Delta a_{e,\text{axial}}$, the truss will remain in the linear-elastic deformation regime. Upon increasing $\Delta a_{e,\text{axial}}$ further, the deformation will become nonlinear due to material plasticity. All tension models will be loaded in such a manner to ensure that they enter the plastic deformation regime, although not up to failure. This section focuses on establishing an overview of tensile deformation including material plasticity, which is required to build the FEA tension models in the upcoming section.

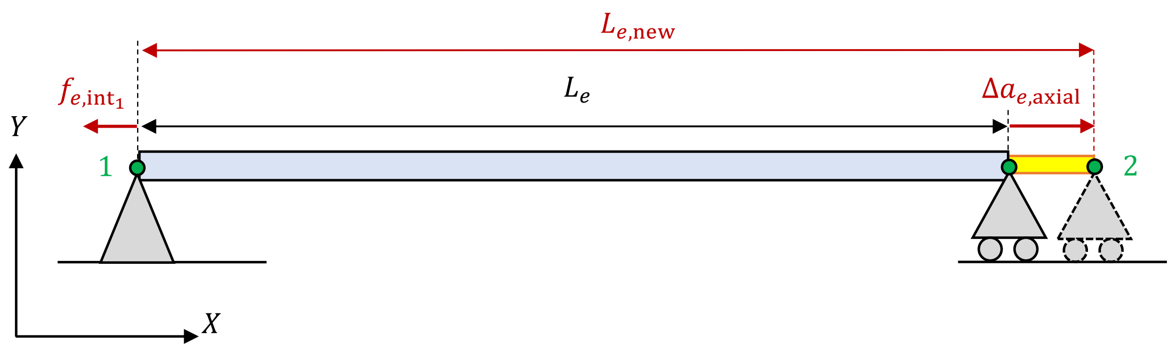


Figure 5.2: Effect of an axial displacement on a truss structure where $\Delta a_{e,\text{axial}} > 0$.

5.2.1. Engineering vs True stress/strain

Specimens under tensile loading are usually modelled through a stress-strain curve. In the linear regime, the structural deformation can be modelled through Hooke's law: $\sigma = E\varepsilon$ where E is the material's Young's modulus. However this linear relation does not hold indefinitely. There comes a point where Hooke's relation no longer holds and the material begins to deform nonlinearly due to material plasticity. From this stage onward, after the specimen is unloaded, it will no longer return to its initial shape. Instead, it will be slightly longer due to plastic deformation ($\varepsilon_{\text{plastic}}$), which can readily be extrapolated from the stress-strain curve using Hooke's law as shown in Figure 5.3. The *yield strength*, σ_{yield} is a first indicator of the structural response departing the linear domain and corresponds to a plastic deformation of 0.2% as shown in Figure 5.3. After this point, the structural

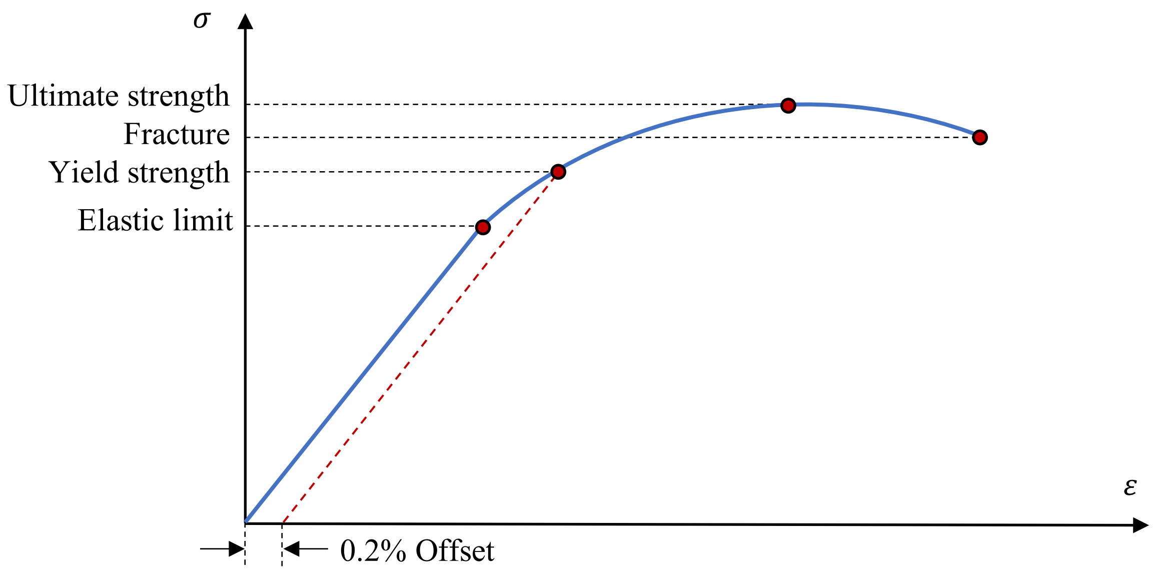


Figure 5.3: Example of a stress-strain curve for a isotropic material subjected to tensile loading.

integrity diminishes, but still continues to increase eventually reaching a maximum known as the *ultimate strength*, σ_{ult}. Hereafter structural integrity diminishes resulting eventually in fracture.

This linear portion of the tensile deformation path for a given specimen is usually simple to estimate. Using the notion established in this section for axial displacement $\Delta a_{e,\text{axial}}$, and internal reaction force at node 1, f_{e,int_1}, Hooke's law can be re-written as follows:

$$\sigma = E\varepsilon \quad \rightarrow \quad \frac{f_{e,\text{int}_1}}{\pi r_e^2} = E\Delta a_{e,\text{axial}} \tag{5.1}$$

$$f_{e,\text{int}_1} = E\Delta a_{e,\text{axial}}\pi r_e^2 \tag{5.2}$$

An important observation at this stage is that the stress σ is determined relative to the original area, and thereby radius, r_e. Models computed in such a fashion are commonly referred to as the *engineering stress/strain* and are denoted as $\sigma^{(e)}, \varepsilon^{(e)}$. Their counterparts are the *true stress/strain* which are a function of the instantaneous change in length (and hence cross-sectional area) and become relevant during plastic deformation when the deformation becomes more substantial. Therefore, the true strain, $\varepsilon^{(t)}$, is computed in the following manner:

$$\varepsilon^{(t)} = \int_{L_e}^{L_e+\Delta L} \frac{1}{L_e+\Delta L}\, dL = \ln\left(\frac{L_e+\Delta L}{L_e}\right) = \ln 1 + \varepsilon^{(e)} \tag{5.3}$$

Where $(L_e + \Delta L)/L_e$ is known as the *extension ratio* and can be directed related to the change in cross-sectional area, A_e. From here the true stress, $\sigma^{(t)}$ can be determined:

$$\frac{L_e+\Delta L}{L_e} = \frac{A_e+\Delta A}{A_e} \quad \rightarrow \quad \sigma^{(t)} = \sigma^{(e)}\left(1+\varepsilon^{(e)}\right) \tag{5.4}$$

The true stress/strain formulation is relevant for the data generation process since the models will be deformed into the plastic regime where relatively important changes in cross-sectional area will affect the stress-strain response of the structure. Although these relationships only hold until the specimens start necking, they will suffice for the data generation purposes where the axial deformation is small. One final important distinction is the relationship between the various types of true strain that are measured. The total strain, $\varepsilon^{(t)}_{\text{total}}$, represents the total strain of the specimen. The elastic strain, $\varepsilon^{(t)}_{\text{elastic}}$, refers to the extrapolated elastic strain; and the plastic strain $\varepsilon^{(t)}_{\text{plastic}}$ is the amount of plastic strain on the material (if the structural deformation has entered the plastic regime).

5.2.2. Building a semi-analytical approximation

Building a fully analytical model to represent tensile plastic deformation is not a trivial process. Extensive experimental testing in conjunction with numerical models are usually required to obtain high-fidelity stress-strain plots including the nonlinear deformation due to material plasticity. However, it is important to have an alternate model which will be used to verify the FEA models that will be hereafter constructed in ABAQUS. When applying a nodal displacement $\Delta a_{e,\text{axial}}$, the structure will elongate to a new length denoted as $L_{e,\text{new}}$. From here, both the engineering and true strain values may be computed in the following manner:

$$\varepsilon^{(e)} = \frac{L_e + \Delta a_{e,\text{axial}}}{L_e} \qquad \varepsilon^{(t)} = \ln\left(\frac{L_e + \Delta a_{e,\text{axial}}}{L_e}\right) \tag{5.5}$$

Assuming that the deformation has entered the plastic regime, the amount of true plastic strain is determined through:

$$\varepsilon^{(t)}_{\text{plastic}} = \ln\left(\frac{L_e + \Delta a_{e,\text{axial}}}{L_e}\right) - \frac{\sigma_{EL}}{E} \tag{5.6}$$

Where σ_{EL} represents the stress at the elastic limit of the material, meaning that $\sigma_{EL}/E = \varepsilon^{(t)}_{\text{elastic}}$. Now consider a function that models the true material stress as a function of the axial displacement, $\Delta a_{e,\text{axial}}$:

$$\sigma^{(t)} = \Theta\left(\Delta a_{e,\text{axial}}\right) \tag{5.7}$$

The true stress can be related back to the reaction force at node 1, f_{e,int_1} while correcting for the fact that the true stress is a function of the immediate cross-sectional area $A^{(t)}$:

$$f_{e,\text{int}_1} = \left[\frac{\pi r_e^2 L_e}{L_e + \Delta a_{e,\text{axial}}}\right] \Theta\left(\Delta a_{e,\text{axial}}\right) \tag{5.8}$$

This ultimately provides a relationship to describe the nodal force at node 1 as a function of induced axial displacement $\Delta a_{e,\text{axial}}$ and the structure's undeformed radius and length: r_e and L_e respectively. The difficulty arises in determining Θ. Given that the chosen material for this project is steel, there exist tabulated values at certain points in the plastic deformation regime, extracted from experimental results. Therefore, Θ can be built as a regression function from experimental data. Provided a dataset of $k+1$ data points, extracted from the plastic deformation regime of steel, a Lagrangian regression can be built leading to:

$$f_{e,\text{int}_1} = \left[\frac{\pi r_e^2 L_e}{L_e + \Delta a_{e,\text{axial}}}\right] \sum_{j=0}^{k} \left[\sigma^{(t)}_j \prod_{\substack{0\le m\le k \\ m\neq j}} \left\{\frac{\ln\left(\frac{L_e + \Delta a_{e,\text{axial}}}{L_e}\right) - \frac{\sigma_{EL}}{E} - \varepsilon^{(t)}_{\text{plastic},m}}{\varepsilon^{(t)}_{\text{plastic},j} - \varepsilon^{(t)}_{\text{plastic},m}}\right\}\right] \tag{5.9}$$

Naturally, this relationship between $\Delta a_{e,\text{axial}}$ and f_{e,int_1} is only valid for when $\varepsilon^{(t)}_{\text{plastic}} > 0$. While the structural deformation is in the linear regime, the expression for f_{e,int_1} reduces to:

$$f_{e,\text{int}_1} = E\pi(r_e - \Delta a_{e,\text{axial}}\nu)^2 \ln\left(\frac{L_e + \Delta a_{e,\text{axial}}}{L_e}\right) \tag{5.10}$$

This approach is not entirely analytical since it utilizes experimental results to build a regression which estimates the plastic deformation of the structure. However, the usage of a Lagrangian regression that interpolates exactly all experimental data and builds a continuous model will suffice as a means to verify the FEA tension models. This will be revisited at the end of the following section, when defining all the FEA parameters for tension data generator. In summary, a proposed semi-analytical model to verify the plastic deformation of truss members is modelled through Equation 5.11.

$$f_{e,\text{int}_1} = \begin{cases} E\pi(r_e - \Delta a_{e,\text{axial}}\nu)^2 \ln\left(\frac{L_e+\Delta a_{e,\text{axial}}}{L_e}\right), & \text{if: } \ln\left(\frac{L_e+\Delta a_{e,\text{axial}}}{L_e}\right) \leq \varepsilon^{(t)}_{\text{elastic}} \\ \left[\frac{\pi r_e^2 L_e}{L_e+\Delta a_{e,\text{axial}}}\right] \sum_{j=0}^{k} \left\{\sigma^{(t)}_j \prod_{\substack{0\leq m\leq k \\ m\neq j}} \left\{\frac{\ln((L_e+\Delta a_{e,\text{axial}})/L_e)) - \frac{\sigma_{EL}}{E} - \varepsilon^{(t)}_{\text{plastic},m}}{\varepsilon^{(t)}_{\text{plastic},j} - \varepsilon^{(t)}_{\text{plastic},m}}\right\}\right\}, & \text{otherwise} \end{cases} \tag{5.11}$$

5.3. FEA Tension model

The finite element approach of solving the tensile problem is relatively straightforward in ABAQUS. The structure is setup exactly as shown in Figure 5.2, except now there is the option to discretize the structure. This will become pertinent when considering mesh convergence, which is relevant when considering the nonlinear deformation regime due to plasticity in the presence of a defect radius, $r_{e,D}$. Recall that the domain of the axial deformation was defined in Chapter 4 as $\Delta a_{e,\text{axial}} \in \pm[0, 0.5L_{e,D}]$ and in turn, the defect length was bounded to 5% of the beam's length: $L_{e,D} = 0.05L_e$. With this in mind, this section aims to establish the settings and data generation strategy of a beam loaded in tension.

5.3.1. Plasticity in ABAQUS

In ABAQUS, material plasticity is defined through the true stress and true plastic strain which were introduced previously. From a stress-strain curve, the nominal stresses and strains can be captured at certain points within the plastic regime, and then converted to true stresses and strains. The data is extracted from experimental tests and built directly into ABAQUS' plastic material database. Table 5.2 shows how the data should be extracted and converted from Figure 5.3. The final inputs into ABAQUS are represented by the green columns.

From this data, ABAQUS is able to extrapolate the material data and apply it in an active FEA simulation thereby modelling plastic deformation. Although this data is not sufficient to capture the

Table 5.2: Data conversion process from nominal to true values from experimental strain values corresponding to steel under axial tension with $E = 210$GPa and $\nu = 0.28$. The green columns represent the final inputs into ABAQUS' plastic material behaviour [4].

Nominal Stress [MPa]	**Nominal Strain** [-]	**True Stress** [MPa]	**True Strain** [-]	**Plastic strain** [-]
200.0	0.00095	200.2	0.00095	0.0000
240.0	0.025	246.0	0.0247	0.0235
280.0	0.050	294.0	0.0488	0.0474
340.0	0.100	374.0	0.0953	0.0935
380.0	0.150	437.0	0.1398	0.1377
400.0	0.200	480.0	0.1823	0.1800

full plastic deformation field of a beam under axial tension, it will suffice to model small amounts of plastic deformation which occur throughout the data generation process.

5.3.2. Mesh convergence and refinement

In an FEA analysis, the user is responsible for discretizing the modelling domain into a numerical mesh. The coarseness of the mesh will ultimately affect the accuracy of the results, though only up to a certain extent. If the mesh is too fine, the model will likely converge and provide reliable results, though the computational cost will be extremely high. On the other extreme, if the mesh is too coarse, then the model might not converge but, if it does, then the reliability would be questionable. Therefore, there is an optimal mesh density for a given model which ensures a high degree of accuracy and model convergence, while remaining computationally efficient. Ideally the mesh should also be refined such that only locations experiencing highly nonlinear deformations and requiring a greater order of computational accuracy have a denser mesh than regions that do not exhibit such deformation. Figure 5.4 shows an example of a mesh refinement procedure for a plate with an open hole. In the case of the tension data generator, mesh convergence studies are required to ensure a computationally efficiency process.

The concept of porosity and material defects during 3D-printing was introduced in Chapter 4. In the case of a truss member, it is represented by a reduced radius at the centre of the truss corresponding to $r_{e,D}$ such that $r_{e,D} < r_e$. In the context of building a FEA model of a beam with such a defect, the mesh is minimally discretized into 3 elements to reflect the region of reduced radius as shown in Figure 5.5. Notice that the numbering of the nodes starts with the boundaries of the structure and then fills-in the remaining nodes. This is the convention that ABAQUS uses when generating an input file and will be adopted throughout this thesis.

Interestingly, since the tensile deformation is only measured at the model's extremities through $\Delta a_{e,i+1_{\text{axial}}}$ and f_{e,int_1}, and overall the deformations are small, mesh refinement does not influence the model's accuracy. This implies that the number of elements can be kept at a bear minimum of 3 or 1 in the case where $r_{e,D} = r_e$. It is numerically verified through ABAQUS for two geometrical configurations. Figure 5.6 shows force-displacement plots for a 30mm beam with a 2mm radius and a

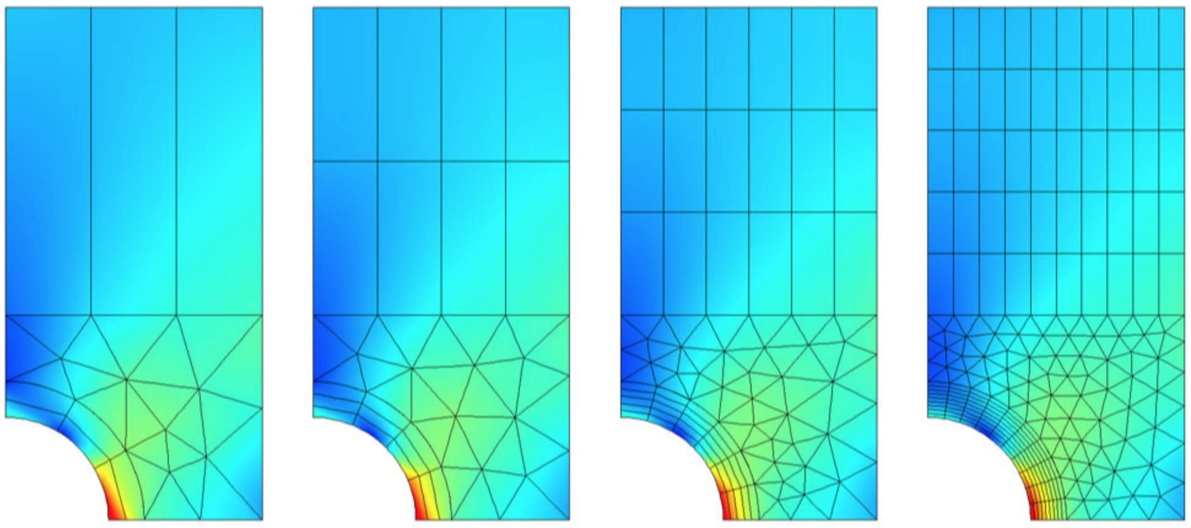


Figure 5.4: Example of mesh refinement for a plate with an open hole using varying sizes of triangular and quadrilateral elements. Note that the mesh density increases closer to the hole to capture the highly-varying stress field [30].

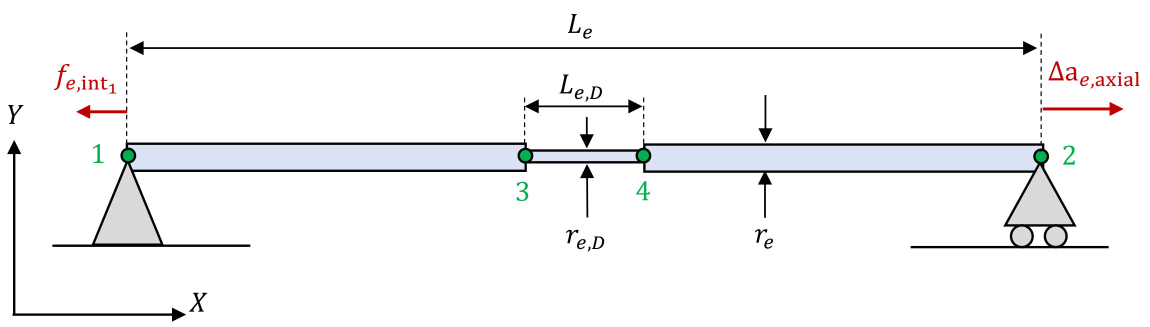
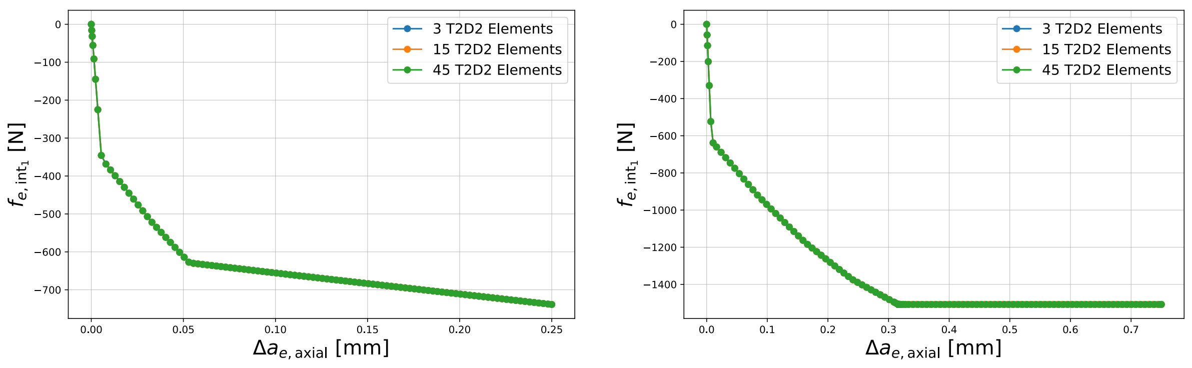


Figure 5.5: Minimal mesh discretization of a beam into 3 elements to include the presence of a structural defect, represented by $r_{e,D}$.

50% defect. The model is meshed with 3 elements (as shown in Figure 5.5), 15 elements (5 elements per truss section), and 45 elements (15 elements per beam section). As may be seen, in both cases, the mesh refinement has no effect on the final plot. Therefore throughout the data generation process, all models will be meshed with 3 elements or 1 in the case where $r_{e,D} = r_e$.



(a) $[L_e, r_e, r_{e,D}] = [10.00\mathrm{mm}, 1.00\mathrm{mm}, 0.75\mathrm{mm}]$ (b) $[L_e, r_e, r_{e,D}] = [30.00\mathrm{mm}, 2.00\mathrm{mm}, 1.00\mathrm{mm}]$

Figure 5.6: Effect of mesh discretization using T2D2 elements on two randomly generated truss members.

5.3.3. Data generation rate

Although mesh refinement is not as critical in the tension case, the numerical step size, is critical. In most cases, it is of importance to ensure the nonlinear numerical solver and FEA model converge. When building data for a neural engine, it is important to consider that the data will be used to train a neural network and any bias or imbalance will result in a less reliable system. Therefore, the step size of the numerical increments should be such that convergence is guaranteed, but also sufficient data is generated.

Initial, minimum and maximum values are chosen for the increment size and will be hereon referred to as Δt_0, Δt_{min}, and Δt_{max} respectively. These values are used when the *Automatic* setting is chosen which essentially allows ABAQUS to decide the increment as a function of structural deformation. The greater the nonlinearity of the path, the smaller the increment and vice versa. This ensures a computationally efficient simulation while maintaining convergence. However, in the case of a data generator, the value of Δt_{max} is extremely important. Consider a beam where L_e, r_e, and $r_{e,D}$ are randomly assigned and equal 30mm, 1.5mm and 1.2mm respectively. Two versions of the same model are configured and Δt_{min} is chosen to ensure numerical convergence in the nonlinear portions of structural deformation. The only difference between the two models is that $\Delta t_{\text{max}} = 0.05$ in one model, and $\Delta t_{\text{max}} = \Delta t_{\text{min}} = 0.0001$ for the other. The latter configuration forces the loading increment to be as small as possible thereby creating a computationally inefficient model. The former provides a substantial amount of flexibility to the solver while still ensuring convergence. The results are displayed in Figure 5.7.

As expected, both models show an identical force-displacement deformation, however the amount of data generated differs vastly. The model with $\Delta t_{\text{max}} = 0.05$ reaches completion in under 10 seconds and generates 36 datasets, whereas the other took around 10 minutes and computed 10,001 increments. Both results are near-identical with the sole difference being an interpolation discrepancy around $\Delta a_{e,\text{axial}} = 0.12$mm where the computationally efficient model does not fully capture the plastic transition. However, the most interesting observations lie at the very beginning of the curve. There is a higher density of data for the model with $\Delta t_{\text{max}} = 0.05$ to allow the numerical solver to capture the deformation. Hereafter, the increments space apart throughout the remainder of the deformation. The only exception is at the first material plastic transition around $\Delta a_{e,\text{axial}} = 0.02$mm where the solver reduces the load step slightly for a few increments to capture the transition. If this data is directly fed into a neural network, the corresponding outputs will produce a more reliable regression at the beginning of the deformation, than throughout the rest of the plot. A simple solution is to produce a uniform amount of data which is manifested by the second case, where $\Delta t_{\text{max}} = \Delta t_{\text{min}} = 0.0001$. In this scenario, the smaller value Δt_{max} forces the numerical solver to produce a uniform amount of data resulting in a highly dense dataset and prevents bias injection into the dataset. This implies that the resultant neural network will be able to model all portions of the structural deformation with the same reliability. While this approach appears ideal, it is not feasible. It takes roughly 10 minutes to generate such a dataset for a single geometrical configuration. If all 2,016 configurations are considered in tension, then the required time to generate the data would be 14 days of non-stop computational processing: assuming, of course, that the process can be automated.

Although possible, the uniform data approach is not suitable since it represents a computationally inefficient system which arguably defeats the purpose of the neural network. Therefore a balance must be found to ensure convergence of the model while producing a sufficient amount of data such that all regions of the deformation curve can be reliably captured. This approach will

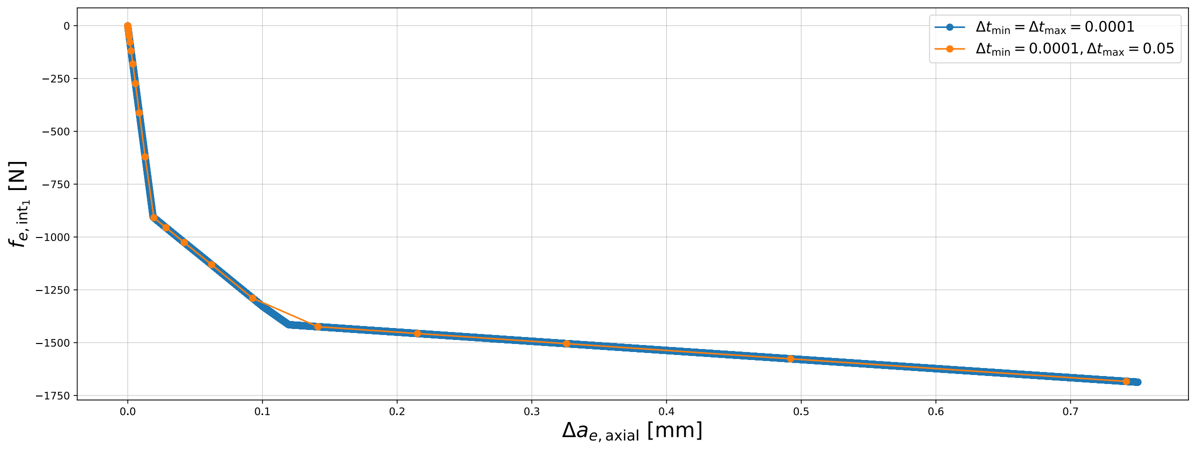


Figure 5.7: Maximum load increment step refinement of f_{e,int_1} vs. $\Delta a_{e,\text{axial}}$ for a randomly generated truss member where: $[L_e, r_e, r_{e,D}] = [30.00\text{mm}, 1.50\text{mm}, 1.20\text{mm}]$.

inevitably introduce some bias within the final dataset, however this can be compensated by the architecture of the neural network. Additionally, the bias in the data is arguably favourable and generates a more robust and expressive neural network in the end [5]. The final step parameters are determined for the extremes of the geometrical boundaries, which are then used for all of the permutations. The chosen step configurations will be displayed along with the remainder settings for the tension generator at the end of this section.

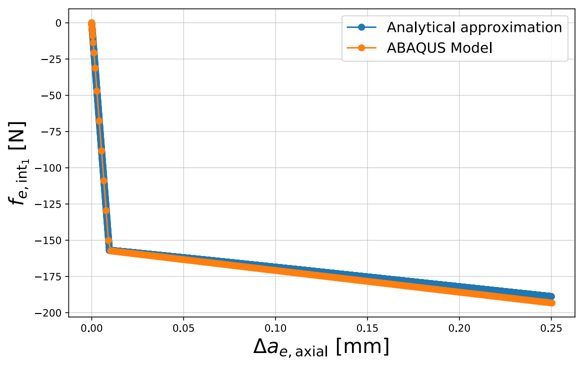
5.3.4. Final ABAQUS parameters

Verification is a necessary step when setting up any engineering model. Unfortunately, the previously developed semi-analytical model does not account for a double-stepped beam and therefore is not suited to model beams with a defect radius where the plastic deformation becomes more complex at the interface between nominal and defective portions of the beam. However, it can be used for the geometrical extremes, where there is no defect present. This leads to four geometrical configurations such that: $[L_e, r_e, r_{e,D}] = [\{10.0, 30.0\}, \{0.5, 2.0\}, \{0.5, 2.0\}]$. The same material dataset used represents the plastic material behaviour of steel that was loaded in ABAQUS from Table 5.2 was used to build the analytical plot define in Equation 5.11. The results are shown in Figure 5.8, and all show an excellent coherence, especially regarding the transition from the elastic to plastic regimes. Slight discrepancies are present at the end of the displacement curve, which likely pertain to either numerical errors or the Lagrangian regression. It is also interesting to note that there is only one point of transition for each model into the plastic regime. Thereby appearing like two linear functions. This only occurs for models with no radial defect. As soon as there is a radial defect present ($r_{e,D} < r_e$), then the transition into and within the plastic regime shows a higher degree of curvature. In any case, this high degree of correlation verifies the boundaries of the numerical tension models permitting us to move forward with the development of the model generator.

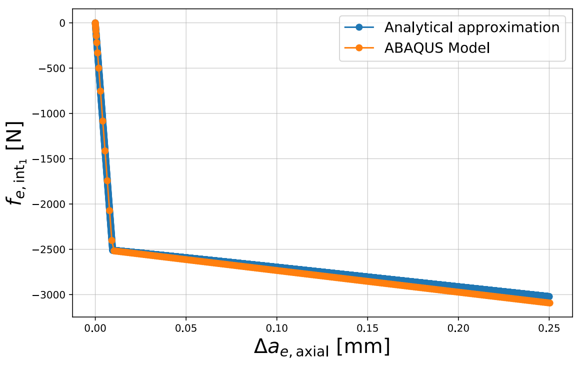
The geometrical boundaries were similarly investigated regarding the data generation and the optimal step sizes were determined such as to ensure model convergence and guarantee a sufficient amount of data production. Given that not all 2,016 models could be individually investigated, a conservative approach is to produce a generous amount of data rather than a sparse (though perhaps sufficient) dataset. All the used parameters used to build the data generator for tension are summarized in Table 5.3. It is important to note that within the numerical solver, the nonlinear geometry has to be enabled to ensure the solver utilizes the true stress/strain values are opposed to the engineering ones. This concludes the tension portion of the model generation. The degree of nonlinearity present is relatively low, though the sharp changes will prove challenging for a neural network to capture. From here, the compression phase of the model generation will be addressed.

Table 5.3: Essential ABAQUS settings for the tension model generator.

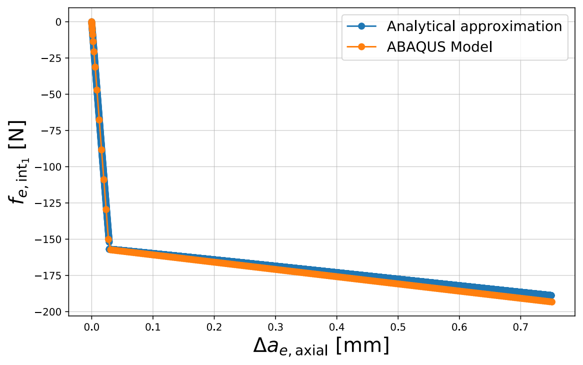
ABAQUS parameter	**Setting**
Element type	T2D2 (2-node linear truss element)
Number of elements	3 (or 1 if $r_{e,D} = r_e$)
Initial increment, Δt_0	0.001
Minimum increment, $\Delta t_{\min}$	0.001
Maximum increment, $\Delta t_{\max}$	0.01
Maximum number of increments	500000
Nonlinear geometry	*Enabled*



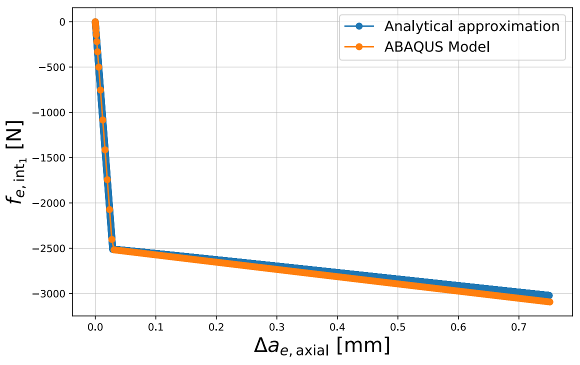
(a) $\{L_e, r_e, r_{e,D}\} = [10.00\text{mm}, 0.50\text{mm}, 0.50\text{mm}]$



(b) $\{L_e, r_e, r_{e,D}\} = [10.00\text{mm}, 2.00\text{mm}, 2.00\text{mm}]$



(c) $\{L_e, r_e, r_{e,D}\} = [30.00\text{mm}, 0.50\text{mm}, 0.50\text{mm}]$



(d) $\{L_e, r_e, r_{e,D}\} = [30.00\text{mm}, 2.00\text{mm}, 2.00\text{mm}]$

Figure 5.8: Nodal reaction force, f_{e,int_1} versus axial elongation $\Delta a_{e,\mathrm{axial}}$ comparing analytical and FEA models for the four truss member configurations which lie on the imposed geometrical boundary.

5.4. Axial compression: an overview

Compression is a nonlinear deformation regime and is an entire field of research on its own. When considering a beam under compression, there are three stages throughout the deformation regime that are of concern. In the beginning, there will be a linear response to the compressive load, similar to that in the case of tension. As the load is increased, there will come a point at which the structure buckles (known as the critical buckling load) and loses a significant portion of its structural integrity by deforming out of the axial plane as shown in Figure 5.9. The final part of the deformation regime is known as the *post-buckling regime* which represents the deformation of the structure after the buckling point. These different regimes are all relevant to the neural element which is required to model the deformation into the post-buckling stage. This section focuses on the data generation process for the compression deformation regime. An analytical model is used to mould the numerical FEA simulation in ABAQUS, which is then used to generate the data for the neural network.

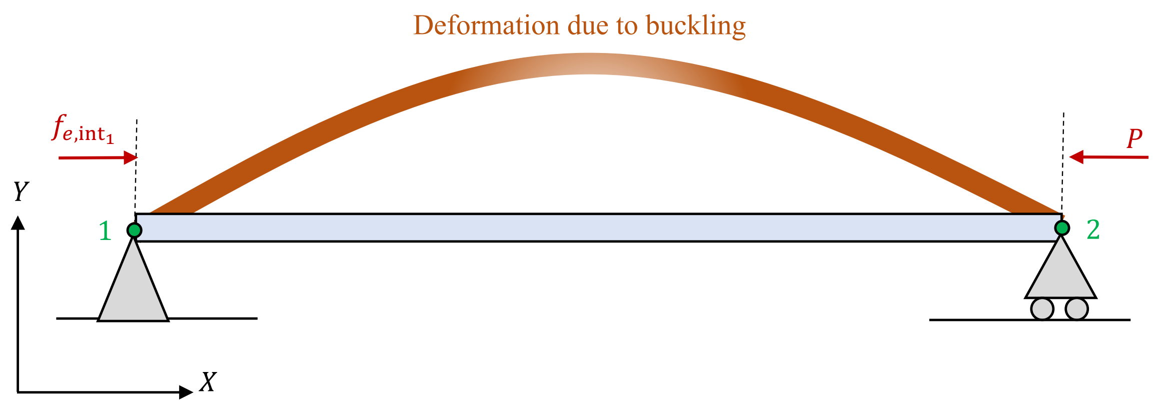


Figure 5.9: Simply supported beam under axial compressive load P. As the axial load is increased, the structure will eventually reach a point where it buckles out of the axial plane (shown in orange).

Arguably, the most important part during the compressive deformation process is the *critical buckling load*, which corresponds to the load required to buckle the structure. Consider the simply supported beam as shown in Figure 5.9. The beam is loaded at the rightmost node by load P. The expression for the total potential energy is shown in Equation 5.12 where w represents the tangential deflection from the beam's nominal position.

$$\Pi = \frac{EI}{2}\int_0^{L_e}\left(\frac{d^2w}{dx^2}\right)^2 dx - \frac{P}{2}\int_0^{L_e}\left(\frac{dw}{dx}\right)^2 dx \tag{5.12}$$

Furthermore, for a simply-supported beam the deflection w can be modelled as follows:

$$w(x) = \sum_{n=1}^{N} a_n \sin\left(\frac{n\pi x}{L_e}\right) \tag{5.13}$$

Combining the two equations together and using the principle of minimum total potential energy yields the expression for the critical buckling load, denoted as P_{cr} which is the lowest buckling load corresponding to a structure buckling in the first mode, hence $n = 1$. This provides the final expression for P_{cr}:

$$\therefore \quad P_{\text{cr}} = \frac{Er_e^4\pi^3}{4L_e^2} \tag{5.14}$$

It should be emphasized that for this approach leads to a deflection solution of $w(x) = 0$ for $P < P_{\text{cr}}$. This shows that in this domain where $P < P_{\text{cr}}$, the material undergoes linear-elastic deformation, just as in the case for axial tension, and therefore the beam remains straight. However, now consider the structure that has an initial out-of-plane deflection w_0. The total potential energy of the structure now becomes:

$$\Pi = \frac{EI}{2}\int_0^{L_e}\left(\frac{d^2w}{dx^2}-\frac{d^2w_0}{dx^2}\right)^2 dx - \frac{P}{2}\int_0^{L_e}\left[\left(\frac{dw}{dx}\right)^2-\left(\frac{dw_0}{dx}\right)^2\right]dx \tag{5.15}$$

Where the initial imperfection may be expressed as:

$$w_0(x) = \sum_{n=1}^{\infty}\lambda\delta_{1n}\sin\left(\frac{n\pi x}{L_e}\right) \tag{5.16}$$

With δ_{1n} being Kronecker-Delta function. Following the same procedure as before, the total potential energy can be expanded leading to:

$$\Pi = \sum_{n=1}^{\infty}\frac{1}{L_e}\left(\frac{n\pi}{2}\right)^2\left[EI(a_n-\lambda\delta_{1n})^2\left(\frac{n\pi}{L_e}\right)^2 - P(a_n^2-\lambda^2\delta_{1n}^2)\right] \tag{5.17}$$

Then the total potential is minimized to obtain an expression for a_n:

$$\frac{\partial\Pi}{\partial a_n} = 0 \quad\rightarrow\quad \sum_{n=1}^{\infty}\frac{2}{L_e}\left(\frac{n\pi}{2}\right)^2\left[EI(a_n-\lambda\delta_{1n})\left(\frac{n\pi}{L_e}\right)^2 - Pa_n\right] = 0 \tag{5.18}$$

$$\therefore\quad a_n = \frac{\lambda\delta_{1n}}{1-\frac{P}{P_{\text{cr}}}} \qquad P\in[0,P_{\text{cr}}) \tag{5.19}$$

Note that the domain of P is such that it cannot equate to the critical buckling load P_{cr}, or else a_n becomes undefined. Using this, the final deflection of the beam while including an initial defect can be derived:

$$w(x,P) = \sum_{n=1}^{N}\left(\frac{\lambda\delta_{1n}}{1-\frac{P}{P_{\text{cr}}}}\right)\sin\left(\frac{n\pi x}{L_e}\right) = \left(\frac{\lambda}{1-\frac{P}{P_{\text{cr}}}}\right)\sin\left(\frac{\pi x}{L_e}\right) \tag{5.20}$$

Where λ represents the amplitude of the initial imperfection or in other words: $\lambda = w_0(x = L_e/2)$. From this expression, the axial shortening can also be directly derived:

$$\Delta a_{e_{\text{axial}}} = \frac{1}{2}\int_0^{L_e}\left[\left(\frac{dw}{dx}\right)^2-\left(\frac{dw_0}{dx}\right)^2\right]dx = \frac{L_e\lambda^2}{4}\left[\left(1-\frac{P}{P_{\text{cr}}}\right)^{-2}-\left(\frac{\pi}{L_e}\right)^2\right] \tag{5.21}$$

Following these derivations, the following summarizes the most important points regarding the buckling and deflection of a simply supported axially loaded beam under compression:

- For a perfect beam with no initial imperfection, there exists a load at which the structure will buckle, called the *critical buckling load*, denoted as P_{cr}.
- In the case of a perfect beam, it is impossible to analytically solve for the deflection w once the structure buckles. This is due to the fact that the neutral equilibrium state is essentially indeterminate.
- For a perfect beam, in the region where $P < P_{\text{cr}}$, the axial deformation complies with the same linear-elastic model derived from Hooke's law in the case for axial tension.
- An imperfect beam is characterized by an initial imperfection manifested by an initial deflection w_0 with amplitude λ.

- The deflection for an imperfect beam can be determined as a function of the applied load P and the axial position x, provided $P < P_{\text{cr}}$. Outside this domain, it is no longer possible to model the structural deflection to the highly nonlinear nature of the post-buckling regime.

It is also important to add that in the case of an imperfect beam, it is very difficult to estimate when buckling occurs, though the previously derived formulation is considered sufficiently valid for small initial defects i.e. λ is very small. These relationships for a beam up until buckling will provide a good basis to verify the developed numerical FEA models. Before proceeding, it is important to consider the other deformation regime during axial compression: the post buckling regime.

5.5. FEA Compression model

Building a compression model in ABAQUS is not as straightforward as tension. Column buckling means that the deformation occurs out of the axial plane, therefore truss elements cannot be used and have to be replaced with beam elements which are able to model bending. Furthermore, a perfect structure modelled under pure compression will not buckle in an FEA model. Either an imperfection or a small moment at one node has to be introduced to initiate buckling. Therefore, it is impossible to build a high fidelity compression model with a singular FEA model. Rather a linear model upon which an eigenvalue analysis is performed has to be analyzed first, and then the output of the linear model is fed into a nonlinear model along with an imperfection. In addition to this, numerical convergence becomes even more important due to the nature of the nonlinear deformation regime. This section focuses on the process to build a singular compression model, and establish the parameters for the entire compression data generator by understanding the parameters affecting the outcome of a post-buckling analysis.

5.5.1. Linear eigenvalue analysis

The first step in creating an FEA compression analysis is building a linear eigenvalue model. This essentially serves to determine the critical buckling load of a beam with no imperfections, thereby using the previously derived relation, which is reiterated below:

$$P_{\text{cr}} = \frac{E r_e^4 \pi^3}{4 L_e^2} \tag{5.22}$$

Within ABAQUS, the model is created as per usual, except when defining the loading step. Rather than initializing a general loading step, a linear perturbation is defined which initiates a buckling analysis once the model is submitted for analysis. Consider one of the geometrical configurations defined in the parameter design space with a L_e = 30.00mm and r_e = 2.00mm. From the above relation, the critical buckling load for this specimen corresponds to P_{cr} = 28930N. From a numerical solver point of view, the buckling load corresponds to the point at which the model stiffness matrix becomes singular. The critical buckling load corresponds to the model's first eigenvalue λ_1. Therefore, the numerical relation which is being analysed is:

$$(\mathbf{K}_0 + \lambda_1 \mathbf{K}_\Delta) v_1 = 0 \tag{5.23}$$

Where v_1 denotes the shapes of the first buckling mode. The linear eigenvalue analysis is not computationally taxing, although mesh refinement is still important. Figure 5.10 plots the difference in critical buckling load versus the analytical solution up to 20 elements. It can clearly be seen that the B22 and B21 elements eventually converge to the same value which corresponds to a critical buckling load of P_{cr} = 28055N, however, there is a noticeable difference with the analysis solution. This is because B21 and B22 elements follow the Timoshenko beam theory which provides a more

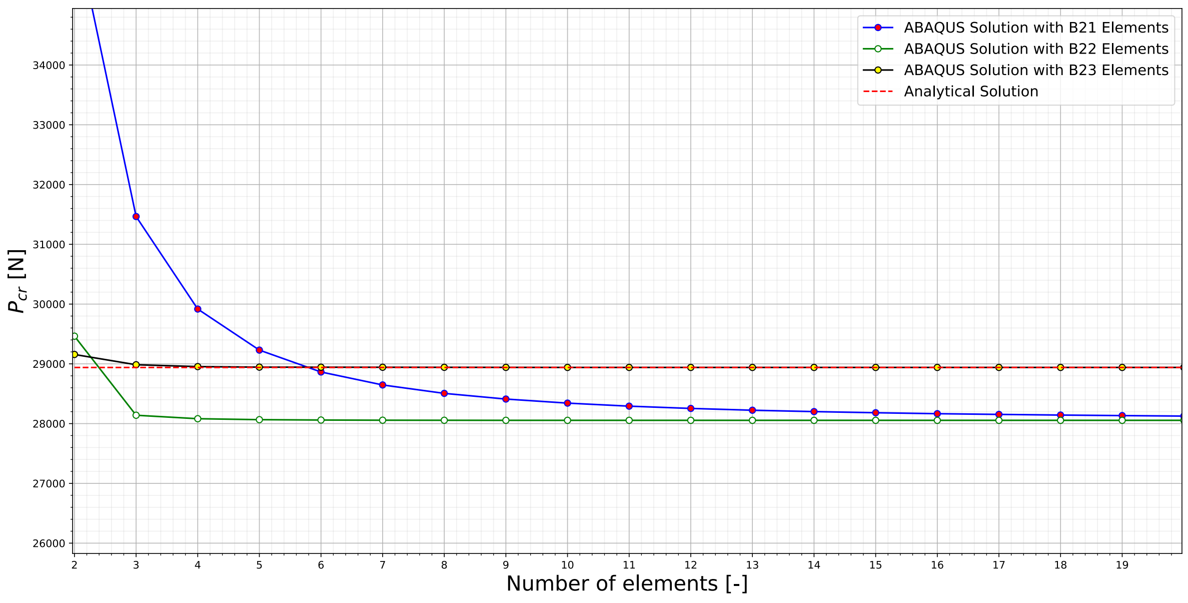


Figure 5.10: Effect of mesh discretization and element selection on the truss member's numerically calculated critical buckling load (P_{cr}) using B21, B22, and B23 beam elements.

conservative estimate of the critical buckling load. The B23 elements on the other hand follow the Euler-Bernoulli beam theory, and therefore converge to the same value as the analytical solution. The conservative estimates from B22 elements coupled with their high degree of modelling accuracy make them an ideal choice for the data generation process. The reason a linear eigenvalue analysis is conducted in the first place is to determine the value of P_{cr} which is then imported into the nonlinear buckling analysis, which will be presented subsequently.

5.5.2. Initiating the nonlinear buckling analysis

After the linear eigenvalue analysis, a nonlinear model is constructed using the same geometrical parameters which extracts some of the features of the linear model, notably the critical buckling load. Form a user point of view, this interaction between linear and nonlinear models happens through the various input files. Figure 5.11 show the process of feeding the linear model into the nonlinear and highlights the most important features.

Prior to conducting the linear eigenvalue analysis, the input file is edited to export the nodal displacements and eigenvalues to a results file. The compression model is then built with the sole difference being the loading step configuration which is defined as a Riks analysis which permits a nonlinear buckling analysis. Then the loading boundary condition is altered to equate to the first eigenvalue from the linear analysis. Finally, prior to initiating the post-buckling analysis, an imperfection has to be introduced via the input file. This imperfection corresponds to the amplitude λ that was introduced in the derivation of the buckling load for a beam with imperfection . ABAQUS requires an imperfection to initiate the buckling analysis, otherwise the result will be pure compression with no axial out-of-plane deformation.

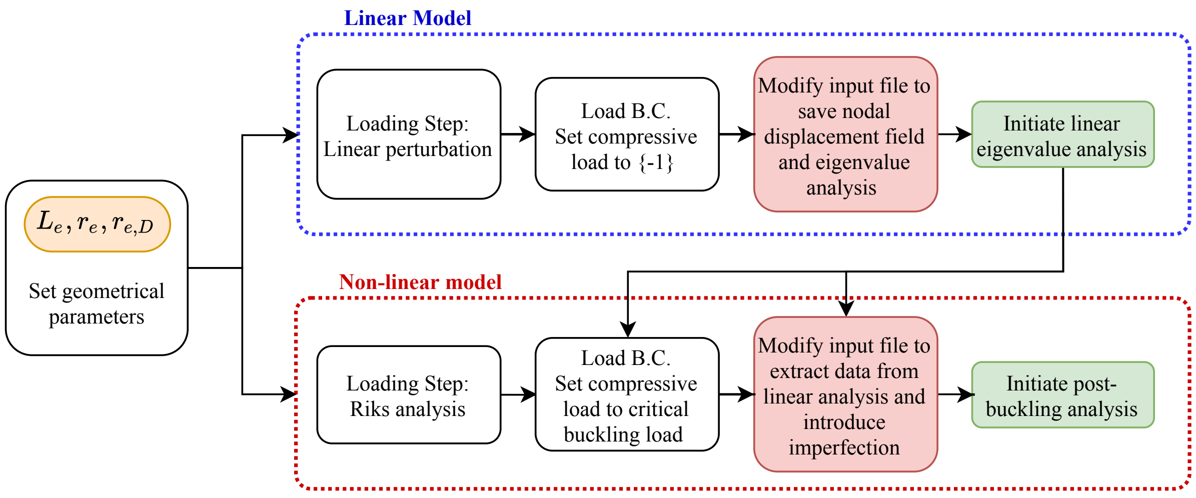

Linear Model
$L_e, r_e, r_{e,D}$
Set geometrical parameters
Loading Step: Linear perturbation
Load B.C. Set compressive load to {-1}
Modify input file to save nodal displacement field and eigenvalue analysis
Initiate linear eigenvalue analysis
Non-linear model
Loading Step: Riks analysis
Load B.C. Set compressive load to critical buckling load
Modify input file to extract data from linear analysis and introduce imperfection
Initiate post-buckling analysis

Figure 5.11: Overview of the interaction between linear and nonlinear models to conduct a nonlinear buckling analysis.

5.5.3. Choosing the initial imperfection

The choice of the initial imperfection is responsible for initiating the buckling analysis and is therefore highly important. Structures with multiple closely-spaced eigenvalues tend to be more sensitive to initial imperfections than others, however this is not the case for a simply supported beam under axial compression. In any case, it is good practice to investigate the effect on the imperfection factor on the numerical analysis to determine how imperfection-sensitive the structure is. ABAQUS defines the initial geometrical imperfection in the following manner:

$$\Delta x_i = \sum_{i=1}^{N} \zeta_i \phi_i \tag{5.24}$$

Where ϕ_i corresponds to the shape of the *i*th eigenmode, and ζ_i is the user-defined scaling factor for the *i*th eigenmode. As a rule of thumb, ζ_i is usually a few percent of the beam's cross-section. Unfortunately, it is not possible to obtain an analytical relation between ζ and λ, however running multiple FEA helps to better understand the effect of the imperfection factor.

Consider a simply supported beam with no defects of length and radius $L_e = 30.00$mm and $r_e = 2.00$mm respectfully. Figure 5.12 shows the effect of the user-defined scaling factor ζ on the nodal reaction force at node 1, f_{e,int_1}. For now, the material plasticity is ignored. The case for $\zeta = 0.00$ represents a beam with no initial imperfection. As expected, the beam does not buckle and continues to deform linearly past the critical buckling load. When $\zeta = 0.01 = 1\%$, the structure buckles almost exactly at the critical buckling load P_{cr}. After the structure buckles (around $\Delta a_{e,\text{axial}} \approx -0.35$mm, the reaction force at the left-most node saturates towards the critical buckling load. Increasing ζ reduces the point at which the structure initially buckles, though for all cases, the nodal reaction forces saturate towards the critical buckling load. It should be emphasized that this observation only holds for the deformation domain that is being used for this project where the axial deformation is capped at 2.5% of the length of each specimen due to the expected small axial deformations. If the axial deformation continues, then the nodal reaction force will continue to slowly increase. Ideally, the value of ζ should be such that buckling is triggered close to the critical buckling load, and the structure deforms linearly until then.

It may also be seen in Figure 5.12 that the analytical solution derived from the Ritz method is also included with an initial imperfection of $\lambda = 0.01$. Unfortunately, it is not representative of the numerical deformation paths, because it does not capture the linear elastic part of the compressive deformation and assumes that the structure buckles instantly. However, the resultant nodal reaction force, f_{e,int_1}, does converge towards the same value as the other FEA models. Furthermore, changing the value of the imperfection factor λ in the Ritz method does not improve the solution. When the applied load is $P = 0$, the axial displacement is proportional to the square of the initial imperfection: $\Delta a_{e,\text{axial}}(P = 0) \propto \lambda^2$. This means that as λ is increased, the axial displacement becomes non-zero even when no load is applied which is representative of the deformation field. Therefore, the Ritz method is only suited for very small initial imperfections and is not able to model the linear elastic deformation of a beam under compression before buckling. However, regarding the numerical FEA models, a scaling factor of $\zeta = 0.01$ is considered suitable to initiate the post-buckling deformation regime.

It bears repeating that the previous discussion on the effect on initial imperfections completely neglects material plasticity, which does play a role in compression. Given that plasticity was included in the tension part of the data generator, it cannot be neglected for compression. If the same beam of length and radius $L_e = 30$mm and $r_e = 2.0$mm, is subjected to compression while

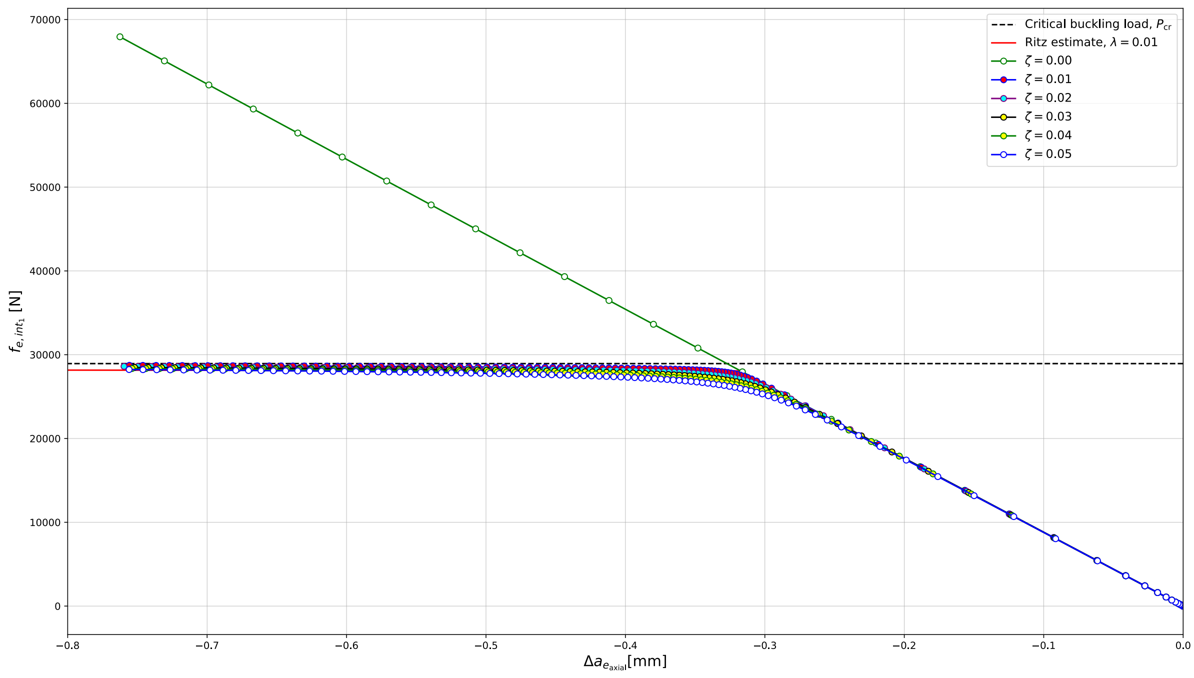


Figure 5.12: Effect of imperfection scaling factors ζ on initiating a post-buckling analysis for a simply supported beam where: $[L_e, r_e, r_{e,D}]$ = [30.00mm, 2.00mm, 2.00mm]. Material plasticity is neglected.

including material plasticity, then the effect of the initial imperfection ζ may be seen in Figure 5.13. The first point of interest is the linear part of the deformation curve which all models have in common. In the case where plasticity was neglected, the linear deformation regime terminated around $\Delta a_{e,\text{axial}} = -0.375$mm. When plasticity is included, this same linear portion ends around $\Delta a_{e,\text{axial}} = -0.025$mm. For the cases when $\zeta = \{0.00, 0.01, 0.02, 0.03\}$, buckling is not triggered and the deformation regime continues to follow a path that is extrapolated from plastic data in the same fashion as tensile plastic deformation. However, for when $\zeta = \{0.05, 0.10, 0.15, 0.20\}$, buckling is triggered and the beam entered the post-buckling regime. It is important at this stage to point out that the load at which the structure buckles is almost an order of magnitude less than the calculated critical buckling load P_{cr}, which emphasises the effect of material plasticity in compression.

One possibility to analytically approximate the critical buckling load of a beam while including material plasticity is known as the Johnson-Ostenfeld approach, and was developed from empirical data collected from column-buckling experiments [95]. The resultant expression relates a material's yield stress σ_y and ultimate stress σ_{ult} to the stress includes from the critical buckling load, denoted as σ_{cr} as shown below:

$$\sigma_{\text{ult}} = \sigma_y \left(1 - \frac{\sigma_y}{4\sigma_{\text{cr}}}\right) \tag{5.25}$$

The expression can easily be altered to correct the critical buckling load to take into account material plasticity, denoted as $\hat{P}_{\text{cr}}$:

$$\hat{P}_{\text{cr}} = \sigma_y \pi r_e^2 \left(1 - \frac{\sigma_y \pi r_e^2}{4P_{\text{cr}}}\right) \tag{5.26}$$

To remain on the conservative side, it is assumed that σ_y represents the engineering stress as opposed to the true stress $\sigma_y^{(t)}$. Therefore, using the same example, a beam of length $L_e = 30.0$mm and

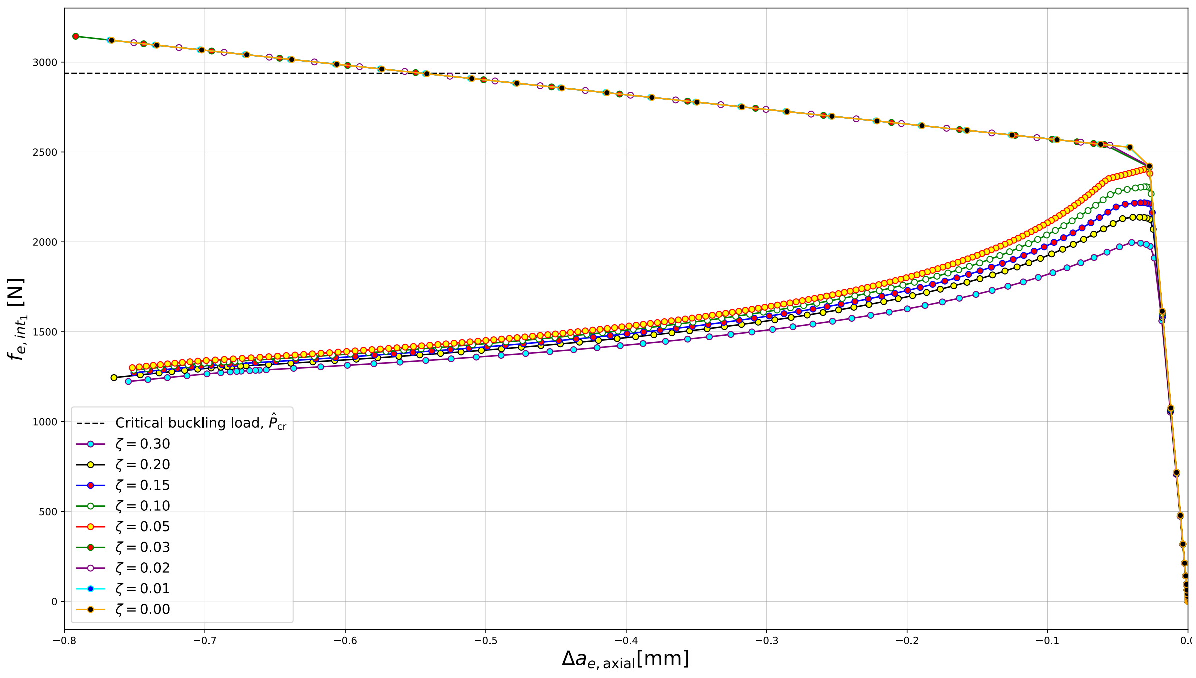


Figure 5.13: Effect of imperfection scaling factors ζ on initiating a post-buckling analysis for a simply supported beam where: $[L_e, r_e, r_{e,D}]$ = [30.00mm, 2.00mm, 2.00mm]. Material plasticity is included.

radius r_e = 2.0mm made form steel has a engineering yield stress of: $\sigma^{(t)}_{\text{yield}}$ = 240MPa. The recalculated critical buckling load is: $\hat{P}_{\text{cr}}$ = 2937N which is shown in Figure 5.13 and is a far better estimate of the critical buckling load as opposed to the original value of $\hat{P}_{\text{cr}}$. The remaining discrepancy between the adjusted critical buckling load and the load at which buckling is initiated ($P \approx$ 2400N) can be attributed to the approximative nature of the Johnson-Ostenfeld formula [95].

Once buckling is initiated, the effect of the initial imperfect has the same effect as the case where plastic material behaviour was neglected. The ideal choice for the FEA model is the lowest possible imperfection factor which initiates post-buckling, while still maintaining the transition point as close to critical buckling load as possible and guaranteeing numerical convergence. For this particular case, this corresponds to a value of ζ = 0.05. It should be noted that ζ = 0.04 is not present in the diagram since it produced a model that diverged entirely, which is thought to be caused by numerical instability between the buckled and nominal-plastic regions.

When considering a defect in the beam such that $r_{e,D} < r_e$, then it is no longer possible to analytically predict the buckling behaviour of the beam using the previously derived relations. This is because the introduction of a symmetrical step in the beam reduces its structural integrity, although not in an intuitive manner. The defect portion will naturally cause greater elastic instability, although it still derives stiffness form the surrounding portion of the beam of greater radius. Creating a comparative (semi) analytical model for post-buckling of a beam with a defected radius is considered out of the scope of this thesis. However, given that the proposed derivations adequately verify the buckling behaviour of a perfect beam with no radial defect, numerical and convergence studies are deemed sufficient for modelling beams with a defect.

It is not realistic to conduct an imperfection sensitivity study for each of the 2,016 geometrical beam configurations. Rather, just as in the case for axial tension, the boundaries of all the geometrical configurations will serve to determine the ζ parameter for the entire compression data generation aspect. Given that all the preferable imperfection scaling factors are relatively similar and of the order of 5%, it is decided that $\zeta = 0.05$ will be adequate for the entire data generation process.

5.5.4. Overview: final ABAQUS parameters

The geometrical configuration of a truss member has a significant impact on the computational efficiency of an FEA analysis. Therefore, a single set of parameters which define the analysis cannot be defined for all models as in the tensile case. This is mostly pertinent to the meshing of the structure as well as the initial imperfection scaling factor ζ to trigger the buckling analysis. Models are therefore analysed per length, and the defined settings are applied throughout a single length domain. Such an approach might be more time consuming than applying a single suite of settings to the entire compression data generator, however ensuring mesh convergence and optimal computational efficiency will be more time efficient in the end. It was found that a total of 15 beam elements, equally divided along a truss member are sufficient to ensure solution convergence. Table 5.4 summarizes the parameters for the compressive data generation process.

With this, the intricacies of the data generator for both the tensile and compressive loading conditions have been addressed. However, manually conducting over 4,000 FEA analyses is simply not feasible meaning that the entire process has to be automated. The following sections focus on the automation of the data generator and how the data repository will be built.

Table 5.4: Essential ABAQUS settings for the compression model generator.

ABAQUS parameter	**Setting**
Element type	B22 (3-node quadratic beam element)
Number of elements	15 (split equally between all 3 truss sections)
Initial arc, Δr_0	0.000001
Minimum arc, Δr_{min}	0.000001
Maximum arc, Δr_{max}	3.000000
Maximum number of increments	500000
Nonlinear geometry	*Enabled*
Imperfection scaling factor ζ	0.05

5.6. Automating the data-generation process

In total, 4,032 models are run in ABAQUS equally split between tension and compression. Manually adjusting the settings of each model is not considered as an option, which is why the entire process is automated. Tension analyses on average are far less complicated than those in compression. As explained previously, for each compression analysis, an additional linear eigenvalue analysis has to be conducted and then fed into the nonlinear model. Furthermore, the computational processing time of the compression models is expected to take far longer due to the severe nonlinearity of the deformation regime and the nature of the Riks solver. Fortunately, automating the entire process is possible thanks to ABAQUS' internal Python terminal. This section is dedicated to outlining the software architecture that was developed to automate the data-generation process.

5.6.1. Software architecture

The global software architecture consists of four separate programs designed to work together to build the global dataset.

1. **The overlord program**: this program contains all the data and parameters that will be used throughout the data-generation process and triggers all underlying programs. It is the sole point of interaction between the user and the data-generation process. All other programs are designed to be left untouched.

2. **Tension model generator**: responsible for building the tension models for analysis.

3. **Compression model generator**: responsible for building the compression models for analysis including the linear eigenvalue analysis.

4. **Data formatter & extractor**: submits the models for analysis and extracts the relevant data from the ABAQUS simulations ($\Delta a_{\mathrm{e,axial}}$ and $f_{\mathrm{e,int_1}}$), and formats the information in an efficient matter for the global data repository.

Graphically, these four programs and their sub-components are displayed in Figure 5.14. Although the code architecture might appear dense, it is simply a combined representation of all the parameters for both the tension and compression cases that have been discussed up until this point. However, there are particular aspects which are worth highlighting.

The sole interface between the user and the software is when defining the input parameters in the overlord program, which can be grouped into three categories: object, FEA, and software properties. Object properties refer to the established parameter design space regarding the truss lengths, nominal and defect radii, as well as the elasto-plastic material properties of steel. The FEA properties refer to the various FEA parameters that were investigated throughout this chapter which affect the numerical result such as the mesh density, the step and loading domain, as well as the initial imperfection scaling factor ζ for the post-buckling analysis. Finally, the software properties refer to the directories where the data will be stored, as well as the hardware distribution in terms of CPU, GPU and RAM.

Once the input parameters are defined, the data generation phase can commence. The process flow for the tension model and compression model is self-explanatory, as well as the entire data extraction phase. However, one additional noteworthy aspect concerns contingency planning. Although the numerical FEA models were analysis and tuned at the boundaries of the parameter design space, some models may still abort the FEA analysis. This could, for example, be due to element distortion in the mesh for particular geometrical configurations or loading increment being too small to satisfy numerical tolerances. Therefore, a *Risk Analysis Report* is embedded into the data extraction phase of the data-generation process. Should a model abort for any reason, it is flagged and written to the risk report along with the summary of the program extracted from ABAQUS' monitor. Whatever data that can extracted from the aborted model is still extracted and saved to the repository, but the fact that it has been flagged will allow the user to instantly review the data and the parameters of the model. This is all done in such a manner such that it does not halt the entire data extraction phase. One the aborted model is flagged and appended to the risk report, the data extraction proceeds with the subsequent model.

At the end of the data generation process, the user will have two global outputs: the data repository and the risk report. The data repository contains the nodal displacements and reaction forces for each model as well as the amount of time it took for the model to be analysed. The risk report is a file which contains a record of all aborted models and the reason why the analysis stopped. Such outputs allow the user to more easily revise the models which were problematic and redefine the input parameters such that they can be revised. Additionally, the CAE models are saved more readily to permit their review at any point in time.

The entire software architecture is designed to reduce the amount of user-interaction when creating the global data repository, thereby also reducing any potential sources of human error. It is important to note that the data automation process does not conduct any analyses on the global data repository, which will be conducted by a separate program described in the following section.

5.6.2. ABAQUS-Python interface: selected notes

From a developer's perspective, there are a few additional attributes concerning ABAQUS' coding interface which are worth noting. Some of these are only pertinent when attempting to simultaneously automate multiple FEA simulations but are relevant for anyone attempting to conduct a similar project to this one.

1. **Python console**: ABAQUS has an embedded Python console which can be used to run scripts and develop models in the ABAQUS environment. However, its latest version only runs on Python 2.7, which should be taken into account if the user wishes to uses modules that may require a more recent version of Python.
2. **Kernel access**: When running scripts, ABAQUS can have unrestricted Kernel access meaning that it can tap into the operating system's underlying hardware. Caution should be exercised since inadvertently abusing Kernel privileges can lead to halting the entire operating system.
3. **Cloud-interfacing**: Many computers use cloud-based drives to save and store data. The time it takes to upload a file to a cloud-based drive depends on the file size and the security measures put into place by the developers. When automating processes through ABAQUS, the software continuously opens, closes, and writes files to its working directory. If the working directory is placed on a cloud-based drive, then it is possible that ABAQUS will abort the entire process because it cannot locate a file in the working directory; this might occur due to the time it requires to sync a file to the cloud drive as compared to the time it takes ABAQUS to re-sample a file. This difference is typically of the order of a few milliseconds but is enough to abort the entire process.
4. **DS files**: Sometimes a .DS_Store file is included in a directory which is used by the operating system's GUI to set the default visual preferences of the directory window. This *ghost* file

cannot be seen by the user when working with a directory, though it can be seen by any form of program which samples a prescribed directory. It is usually recommended to work around this file when designing a program such that it does not cause any errors. This issue is particularly known to affect Apple users.

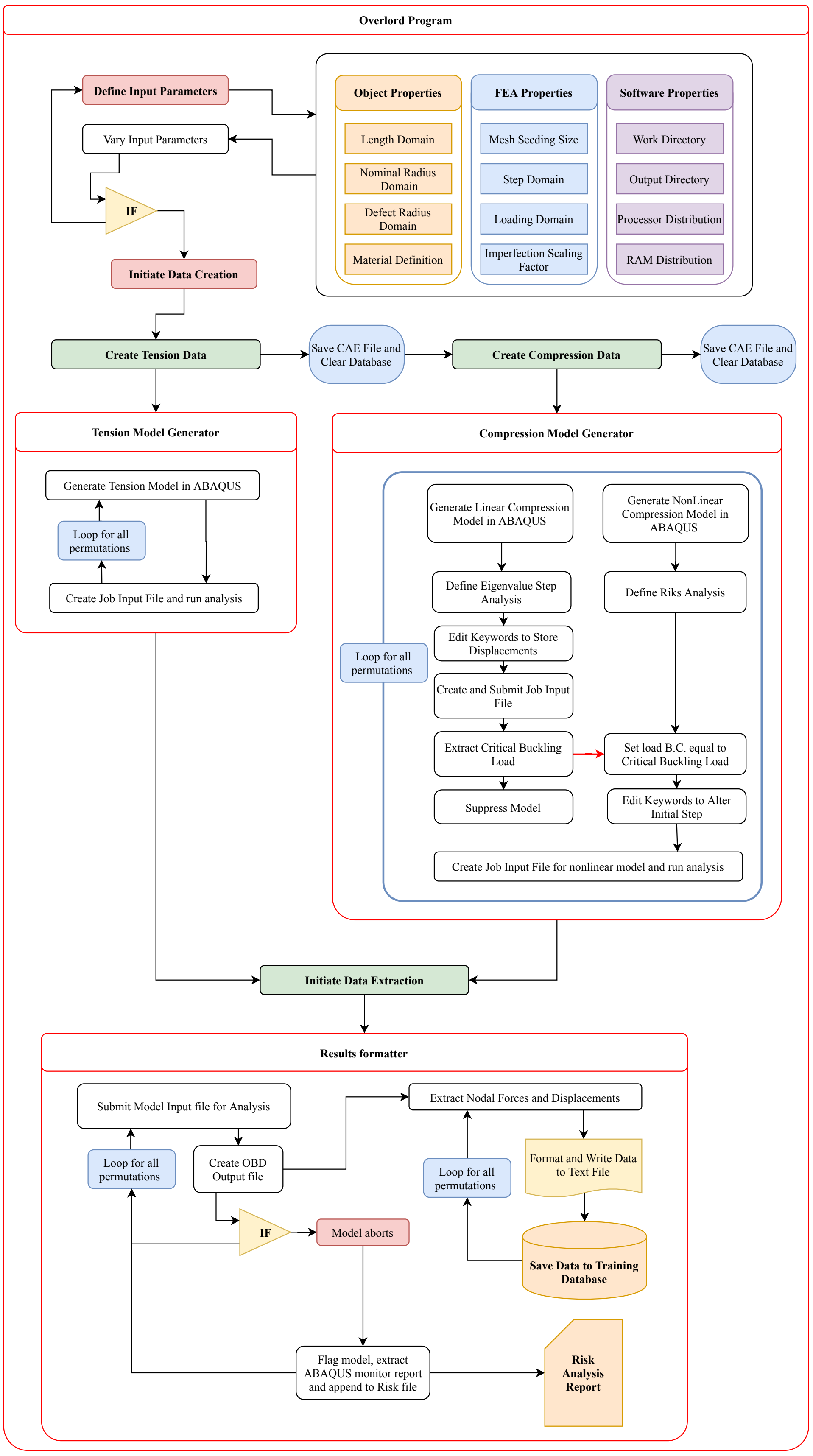

Overlord Program
Define Input Parameters
Vary Input Parameters
IF
Initiate Data Creation
Object Properties
Length Domain
Nominal Radius Domain
Defect Radius Domain
Material Definition
FEA Properties
Mesh Seeding Size
Step Domain
Loading Domain
Imperfection Scaling Factor
Software Properties
Work Directory
Output Directory
Processor Distribution
RAM Distribution
Create Tension Data
Save CAE File and Clear Database
Create Compression Data
Save CAE File and Clear Database
Tension Model Generator
Generate Tension Model in ABAQUS
Loop for all permutations
Create Job Input File and run analysis
Compression Model Generator
Generate Linear Compression Model in ABAQUS
Generate NonLinear Compression Model in ABAQUS
Define Eigenvalue Step Analysis
Define Riks Analysis
Edit Keywords to Store Displacements
Loop for all permutations
Create and Submit Job Input File
Extract Critical Buckling Load
Set load B.C. equal to Critical Buckling Load
Suppress Model
Edit Keywords to Alter Initial Step
Create Job Input File for nonlinear model and run analysis
Initiate Data Extraction
Results formatter
Submit Model Input file for Analysis
Extract Nodal Forces and Displacements
Loop for all permutations
Create OBD Output file
Loop for all permutations
Format and Write Data to Text File
IF
Model aborts
Save Data to Training Database
Flag model, extract ABAQUS monitor report and append to Risk file
Risk Analysis Report

Figure 5.14: Code architecture designed to automate the data-generation process.

5.7. Closure

The final data repository includes all the data gathered from the tension and compression data generators. This implies that for a given geometrical configuration, the entire deformation field spanning from tension to post-buckling including material plasticity can be generated for 2,016 geometrical configurations. Figure 5.15 shows an example of two full-deformation curves for a specimen, one for a perfect beam, and the other for the same beam with a 50% radial defect. The impact of the structural integrity is significant. The load-carrying capability of the structure with a 50% radial defect is reduced by nearly 75% when compared to that of the intact structure. Figure 5.15 also serves to reinforce the manner in which the data is stored in the repository. It is important to understand that the amount of data is defined by axial displacement ($\Delta a_{\text{e,axial}}$), rather than the geometrical designation. For a given length, radius, defect radius and axial displacement ($L_e, r_e, r_{e,D}$ and $\Delta a_{\text{e,axial}}$), there is a a singular value for the nodal reaction force $f_{\text{e,int}_1}$. This is referred to as a single *dataset* from here onwards.

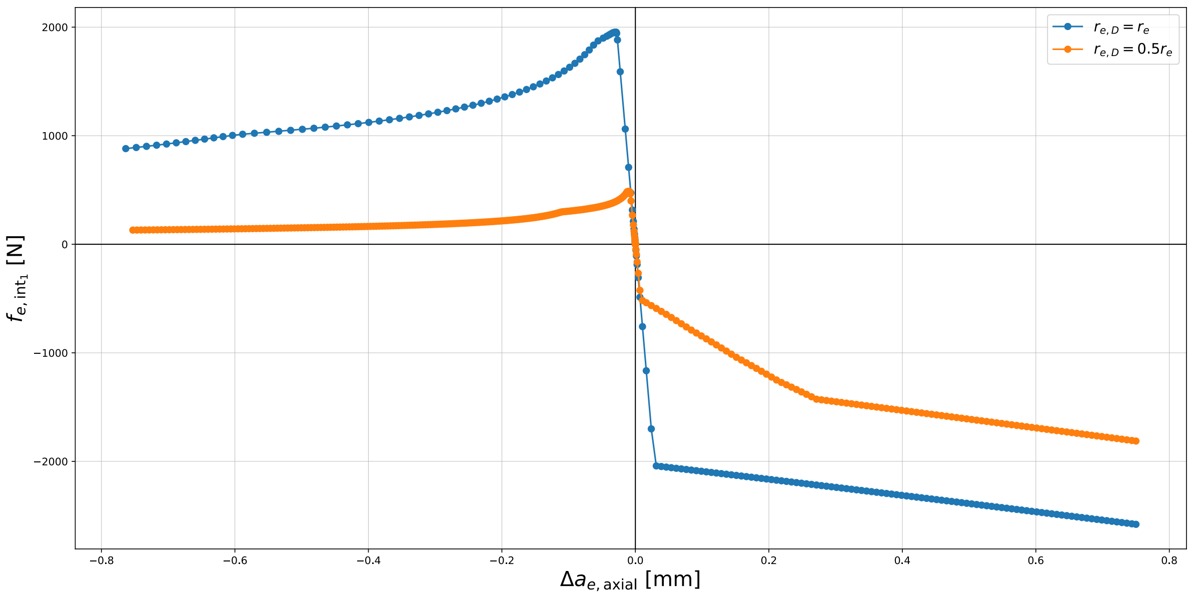


Figure 5.15: Example of the full deformation domain including material plasticity and post-buckling within the prescribed boundaries for specimen of length and nominal radius: $L_e = 30.00$mm, $r_e = 1.80$mm. All the defective configurations within the parameter design space are considered.

Roughly 5% of the generated FEA models aborted due to errors and were filed in the risk report due to convergence issues, which pertained to the compression models. These were manually adjusted to achieve a converged result. The global data balance between tension and compression is 40%:60% respectively meaning that the data repository is biased towards compressive behaviour. However, given the amount of data available, it is unlikely that this offset will cause any problems during the training phase of the neural network. Overall, the automated generator built and ran 4,032 FEA models equally split between tension and compression resulting in 587,142 datasets; more than 5 times the amount initially anticipated. This provides a solid foundation upon which a neural network can be developed and trained.

6
Building the Neuromorphic Engine

With the data repository fully constructed, the neural network training procedure can begin. Chapter 1 noted that one of the goals of this thesis is to develop the neural network in a non-conventional manner. This mainly refers to the architecture of the network and the choice of hyperparameters. These values are conventionally determined through a trial-and-error approach, which is a highly inefficient process. Although it might be suitable for regression problems on a smaller scale, a trial-and-error approach is inadequate for the level of expressiveness that is demanded from the neural network in this thesis. As a reminder, the trained network will have to model the following deformation regimes of an axially loaded structural member:

- Axial tensile deformation including material plasticity.
- Axial compressive deformation including post-buckling and material plasticity.

The amount of nonlinearity present in each case is significant, and when considering the domain of the parameter design space, the scale of the regression problem increases greatly. Choosing the appropriate neural architecture is therefore of paramount importance and cannot be left to a trial-and-error process. Rather, the neural network will be programmed to choose its own neural architecture and hyperparameters. The sole requirement from the user will be to specify which hyperparameters to consider, which is essentially equivalent to choosing the parameter design space of the neural network.

Providing a neural network with the ability to alter and design its own architecture, based on the type of data flowing through it, is suggestive of the morphable neural network that was introduced in Chapter 2. Although the morphable neural network of Chapter 2 revolves around the Dropout method (a tool used to alter a neural network's architecture by removing nodes and layers during the training process), the concept of morphability is a starting point for the contents of this chapter. Hence, to reflect its morphable nature, the neural network developed here is renamed to account for its ability to alter its own settings and neural architecture as a neuromorphic engine. This chapter focuses on the process of building and designing it. Many of the terms used in neural network design were outlined in Chapter 2 and will not be reiterated here.

6.1. Data formatting & preparation

The first step in training any neural network is the proper formatting of the data. The global data repository developed in the previous chapter stores data in terms of specimen length, nominal radius, defect radius, axial displacement, and corresponding nodal reaction force. Prior to feeding the data through the neuromorphic engine, it has to be properly formatted to ensure a smooth training process. This consists of two main parts: data normalization and data segregation.

6.1.1. Data normalization

The first step is normalizing the entire data repository to a uniform scale. The importance of data normalization cannot be overemphasized and is often overlooked when tailoring data for a neural network. Typically, a data repository is normalized such that all datasets fall within a highly constrained domain such as $[0,1]$ for example. If the data were not normalized, then large differences in values would either saturate or crash the backpropagation scheme which hampers the learning process. Users who manage to achieve a properly trained neural network without normalizing the data are often considered lucky. By restricting all data to a small confined domain, the backpropagation scheme is able to maintain its efficiency.

In the current case, normalizing the entire data repository is a relatively simple process. Once the boundary values are known for all parameters, then every single data point can be normalized. Although this could work, the complexity of the deformation regime that the neuromorphic engine is required to capture cannot be overlooked. It is worth repeating that both tensile and compressive deformation for any geometrical configuration have to be modelled by the final neuromorphic engine. Although the values of L_e, r_e, and $r_{e,D}$ will always be positive, the displacements and nodal forces ($\Delta a_{e,\text{axial}}$ and $f_{e,\text{int},1}$) fluctuate between positive and negative values. If these were also to be normalized in a $[0,1]$ domain along with the geometrical parameters, then it will be all the more difficult for a machine to differentiate the compressive and tensile cases. One simple way to help the neuromorphic engine learn the data trends and deformations is to provide it with the following simple guidelines:

- If $\Delta a_{e,\text{axial}} > 0$, then $f_{e,\text{int},1} < 0$ (Tensile case).
- If $\Delta a_{e,\text{axial}} < 0$, then $f_{e,\text{int},1} > 0$ (Compressive case).

These are simple properties of the deformation domains which require tailored normalized data domains. Therefore, positive quantities for either $\Delta a_{e,\text{axial}}$ or $f_{e,\text{int},1}$ will be normalized to $[0,1]$ and negative ones will be normalized to $[-1,0]$. Guidelines such as these can make a significant amount of difference with regard to the expressivity of the final product.

Table 6.1: Data normalisation boundaries for all parameters flowing into and out of the neural network. Recall that L_e, r_e, $r_{e,D}$ and $\Delta a_{e,\text{axial}}$ are inputs to the network and $f_{e,\text{int},1}$ is the projected output.

Parameter	**Max boundary value**	**Min boundary value**	**Normalization domain**
L_e	30.0mm	10.0mm	$[0,1]$
r_e	2.0mm	0.5mm	$[0,1]$
$r_{e,D}$	r_e	$0.5r_e$	$[0,1]$
$\Delta a_{e,\text{axial}}$	0.75mm (*Tension*) 0.00mm (*Compression*)	0.00mm (*Tension*) -0.88mm (*Compression*)	$(0,1]$ if: $\Delta a_{e,\text{axial}} > 0\text{mm}$ $[-1,0)$ if: $\Delta a_{e,\text{axial}} < 0\text{mm}$ 0 if: $\Delta a_{e,\text{axial}} = 0\text{mm}$
$f_{e,\text{int},1}$	0.0N (*Tension*) 2503.3N (*Compression*)	-3185.1N (*Tension*) 0.0N (*Compression*)	$(0,1]$ if: $f_{e,\text{int},1} > 0\text{N}$ $[-1,0)$ if: $f_{e,\text{int},1} < 0\text{N}$ 0 if: $f_{e,\text{int},1} = 0\text{N}$

With all this in mind, Table 6.1 summarizes the data normalization domains of all the parameters that will be flowing into and out of the neuromorphic engine as well as their respective boundary values. With this the full data suite falls within $[-1, 1]$, which is optimal for network training.

6.1.2. Data segregation

Once the data has been normalized, the last step in the formatting process is splitting it into training, validation, and testing data. As outlined in Chapter 2, a good rule of thumb for these ratios is 50%, 25%, and 25% respectfully. However, in practice, these ratios are only effective for a small data repository. When data quantity is not a constraint, than these ratios can be significantly altered in favour of the training ratio. In any case, the ratios for the three categories will remain: $\{n : (100 - n)/2 : (100 - n)/2\}$ where $n \in [50\%, 100\%)$.

For the current topic, n is chosen to be 98%. This means that 98% of the data repository will be randomly sampled and used to train the network. Of the remaining 2%, half will be used to validate the training process of the network after each epoch, and the other half will be held out for testing the network once fully trained. Given that the amount of datasets in the repository is in excess of half a million, these ratios are deemed sufficient to properly evaluate the training process of the network.

6.2. Morphability: A self-designed neural architecture

Designing an environment that enables a neural network to define its own hyperparameters is not as daunting as it may seem. The end goal is to almost entirely remove the human element from the process thereby eradicating the trial-and-error approach, which is often used when building a neural network. The trial-and-error process is generally only applicable in the case where the complexity of the problem is either limited (which is a subjective concept in itself), or there is a foundation to the problem in literature which has already been addressed. Unfortunately, the current problem does not conform with either of these scenarios: the degree of complexity is considerable since the final neural network has to be able to model both tensile and compressive deformation. In addition, the amount of nonlinearity present is considerable, caused by material plasticity and/or the post-buckling regime. If a neural network had to be built by trial and error for this problem, then a justifiable reflex would be to build an extremely deep network with many nodes and hidden layers; since the deeper the network, the greater the degree of expressivity. As shown in Chapter 2, this is not necessarily the case. The relative importance of hyperparameters cannot be generalized and differs from problem to problem. Therefore, a trial-and-error approach is not desired.

Removal of the human element from the neural architectural design can be accomplished by using an optimizer environment based on random selection, with some very strict parameters. The only role of the user is to specify which hyperparameters to investigate, the random sampling method used, and which metric to target. Given the specifications the algorithm is processed in the following fashion:

1. The algorithm chooses a set of hyperparameters within the prescribed boundaries and builds a neural network.
2. It then randomly samples a portion of the data repository (typically 5-10%) using the defined random sampling method.
3. It then trains the network within a highly limited number of epochs (usually less than 100).

4. Throughout the training process, it monitors the metric defined at the very beginning.
5. If the algorithm observes no improvement of the defined metric over 5 consecutive iterations, then it halts the training process. Otherwise, it continues to train the network up until the limit number of epochs.
6. After training, it saves the data and builds another neural network with another set of hyperparameters and repeats the entire process.
7. Once finished, the best hyperparameter configuration is selected which corresponds to the network that displayed the best rate of improvement in the defined metric, over the fewest number of epochs.
8. The user is provided with a high-fidelity neural network architecture at a low computational cost. The selected neural network is trained over the entire data repository.

The random sampling method is a crucial choice. If only a fraction of the data repository is to be sampled each time, then it could be that the random sampler extracts data from only the tensile or compressive case. However, given that the algorithm re-samples the data repository for each network configuration, there is a certain reliability in the results after many evaluations. Although this alone is not sufficient, since the amount of risk in generating a final network that completely disregards a deformation regime is too high. An more reliable approach is not to enable pure random sampling. Latin Hypercube and Improved Latin Hypercube are both proven techniques for randomly sampling data by subdividing the data repository into clusters of similar data, and then randomly sampling the data within each cluster equally. This ensures that a representative random group of data is used for the training process since it removes the risk of sampling within a confined domain. A review of Latin Hypercube's capabilities and its improved version may be found in [128] and [31] and will not be discussed here.

Another aspect which has to be addressed, is the extent of the hyperparameter space the algorithm investigates. If all permutations within the prescribed boundaries are investigated, then it could easily be argued that the algorithm is no better than an automated version of the trial-and-error approach. Thankfully, this is not the case. The developed algorithm tailors its hyperparameter choice depending on the improvement in the defined metric of the previous iteration, and the overall best configuration to date. If it experiences continuous diminishing returns over the course of 5 consecutive network configurations, then the process is automatically ended since the optimal solution has been determined. In practice, less than 20% of all permutations are investigated.

It could be argued that since not all permutations are considered, then the end product will never be as good as it could be. Although valid, it is important to recall that there is no such thing as an infallible machine, or an infallible human. An intelligent entity is not infallible. Leaving room for errors and discrepancies has proven to create more robust neural networks, which is why a user should not be obsessed with finding the *perfect* neural architecture. There are two other areas that merit explanation before initiating the algorithm: the choice of metric and the hyperparameters to be optimized.

6.2.1. R-squared Metric

Despite the overall complexity, the problem at hand is simply a regression problem. Most available metrics for neural networks outlined in Chapter 2 pertain to selection-based problems. For regression cases, the R-squared metric (also known as the *coefficient of determination*, or R^2) is a measure of how well a regression fits a dataset. It is computed as a ratio between the regression sum of squares and the total sum of squares between a regression projection and a dataset. Mathematically, it is expressed as:

$$R^2 = 1 - \frac{\sum_i (f_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} \tag{6.1}$$

An R^2 value of 1 indicates a perfect fit since it means that $\sum_i (f_i - \bar{y})^2 = 0$. Given the amount of data available, a network displaying acceptable performance has an associated R^2 value of 0.9 or greater. In addition to the R^2 metric, the algorithm also observes the mean squared error loss function as a backup, which is defined as:

$$\mathscr{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 \tag{6.2}$$

Even though the loss function is evaluated after a batch sweep through the network, the two metrics should be correlated such that as R^2 nears closer to 1, the mean-squared error ($\mathscr{L}_{\text{MSE}}$) approaches 0. Should this not be the case, the algorithm will flag this problem and allow the user to intervene.

6.2.2. Hyperparameter selection

Prior to initiating the developed optimizer algorithm, the hyperparameters selected to be optimized are chosen and summarized in Table 6.2. The neuron density and number of hidden layers are obvious choices which always affect the expressiveness of the trained neural network. Additionally, the activation function used in the hidden layers will be interchanged between the ReLU (rectified linear unit), which has proven itself as a piecewise numerical function that ensures an effective training scheme; and the ELU (exponential linear unit). The ELU is a continuous version of ReLU, similar to the leaky ReLU, to prevent saturation of the backpropagation scheme. Furthermore, the learning rate of the optimizer (ADAM) is also considered. Lower learning rates promote a steadier training progression whereas higher ones promote severe fluctuations. However, higher learning rates do guarantee a faster convergence, which is appealing when considering the size of the data repository. The Dropout rate is also implemented to prevent feature co-adaptation ranging from 0.00 (not activated) to 0.50. In the case of a deep network, higher values of dropout are favorable to ensure regularization. Since the depth of the network is not certain however, the dropout rate cannot be easily defined which is why it is another hyperparameter variable. Finally, the affect of the batch size is also considered, which varies between 2,000 and 50,000 datasets.

In addition to the hyperparameters that are varied, there are fixed hyperparameters. Table 6.3 shows those that are fixed throughout the entire process. As mentioned previously, the number of epochs is set relatively low to 200. However, it is expected that the algorithm will never reach this

Table 6.2: Chosen hyperparameters to optimise.

Hyperparameter	**Symbol**	**Min boundary value**	**Max boundary value**
Neuron density	ρ_{neuron}	5 neurons per layer	1,000 neurons per layer
Hidden layers	n_{hidden}	0 hidden layers	100 hidden layers
Activation function	Φ_i	ReLU	ELU
Learning rate	L	0.0005	10.0000
Dropout rate	D	0.00	0.50
Batch size	V_{batch}	2,000 datasets	50,000 datasets

Table 6.3: Fixed hyperparameters throughout the optimisation algorithm.

Hyperparameter	**Symbol**	**Constraint**
Epochs	n_{epoch}	200
Network shape	*n.a.*	Brick
Optimizer	*n.a.*	ADAM
Loss function	$\mathscr{L}$	Mean-squared error (MSE)
Output activation	Φout	Hyperbolic tangent
Bias	b_i	*Enabled*

number due to the cutoff constraint: if the improvement in the R^2 metric over 5 consecutive epochs is less than 0.1%, then the process terminates and proceeds to the next configuration. The network shape is an interesting parameter which as of now is constrained to a *brick* shape. This means that the number of neurons in each layer is the same. As described in Chapter 2, there are significant advantages to alerting the network shapes, which expand and compress the dimensionality of the data. Its effect will not be investigated in this thesis. The optimizer is chosen as ADAM due to its consistent performance and reliability, while ensuring a fast convergence of the network. The MSE loss function was chosen previously and is a suitable evaluation for regression-type problems. The output activation Φ_{out} is perhaps the most important parameter and is selected to be the hyperbolic tangent for multiple reasons. Its continuous natures permits a consistent evaluation of the derivative when starting the backpropagation process, and it generates an output the range of $[-1,1]$, which is required since the nodal reaction forces, f_{e,int_1} were normalized to $[-1,1]$. Finally, bias is enabled throughout the entire process.

This concludes the overview of the hyperparameters used throughout the process of optimizing the neural architecture. From here, the algorithm may be enabled to determine a suitable suite of parameters. However, before proceeding, this is a good point to review the differences between datasets, batch and epochs.

- **Dataset**: Single set of values in the repository $\{L_e, r_e, r_{e,D}, \Delta a_{e,\text{axial}}, f_{e,\text{int}_1}\}^{(i)}$.
- **Batch**: One batch contains n datasets. The synaptic weights of the network are updated through backpropagation after each batch (and not each dataset).
- **Epoch**: One epoch is one pass of the entire data repository through the neural network.

6.2.3. Chosen architecture

The number of possible neural architectures within the defined hyperparameter boundaries amount to 400,000. Out of those 400,000, the developed algorithm only tested 5,824 before coming to a stop. This corresponds to roughly 1.5% of the total amount of neural architecture permutations. Table 6.4 summarizes the performance metrics of the algorithm while Table 6.5 shows the hyperparameters which the algorithm considered to be the best possible combination for developing a neural network on the data repository. What is remarkable is that the selected neural architecture experienced a performance saturation within 17 epochs which is incredibly fast. This means that over those 17 epochs, the R^2 performance had already saturated to a value of 0.9989 by epochs number 12 (since the program assumes convergence if the performance improvement over 5 consecutive epochs is less than 0.1%). The total run-time of the algorithm was around 42 hours. Although this might seem like a long time, devoting less than two days to determining an optimal hyperparameter configuration for a neural network is very short.

The developed algorithm has shown that a neural network with 5 hidden layers each containing 100 neurons is sufficient to model the nonlinear tensile and compressive deformation regimes of 2,016 different truss members. The algorithm also showed that there is no need to considered every network configuration since many optimums exist. From the chosen hyperparameters, the network can be trained on the full data repository.

Table 6.4: Performance metrics from the developed algorithm which selects an optimal configuration of hyperparameters on a randomly sampled dataset from the global repository.

Entity	**Value**
Total number of potential neural architectures	400,000
Number of neural architectures tested	5,824
Time required to reach completion	≈ 42 hours
Average training time per network	≈ 26 seconds
Epochs before performance saturation of best network	17
Attainable R^2 of best network	0.9989

Table 6.5: Selected hyperparameters.

Hyperparameter	**Symbol**	**Final value**
Neuron density	ρ_{neuron}	100 neurons per layer
Hidden layers	n_{hidden}	5 hidden layers
Activation function	Φ_i	ReLU
Learning rate	L	0.0008
Dropout rate	D	0.00
Batch size	V_{batch}	15,000 datasets

6.3. Training the neuromorphic engine

The chosen neural architecture for the neuromorphic engine was outlined in the previous section and can now be fully trained on the entire data repository. This section focuses on assessing its capabilities in terms of general performance and compounded error analysis. The assessment is an important step in locating any shortcomings or limitations of the neuromorphic engine, since this is the last stage where it can still be corrected. Should there be limitations which cannot be solved, then it is important to be aware of them before moving forward.

6.3.1. General performance

Once unleashed on the entire data repository, the neuromorphic engine is trained across more than half a million datasets representing the tensile and compressive deformation regimes for the selection of truss members. The training process for the hyperparameter optimization was near-identical to the hyperparameter optimization algorithm, with the exception of the early-stopping. In the previous section, the training process was halted if the improvement in the R^2 metric over 5 consecutive epochs was less than 0.1%. During the training process of the chosen configuration, the early-stopping criteria is slightly adjusted to halt the process not when the level of accuracy saturates, but if the R^2 improvement inverts and starts depreciating. This is a phenomenon that sometimes occurs during the development of an intelligent system: there is an optimal number of epochs to the train the model before its performance substantially decreases. Although this pertains more to

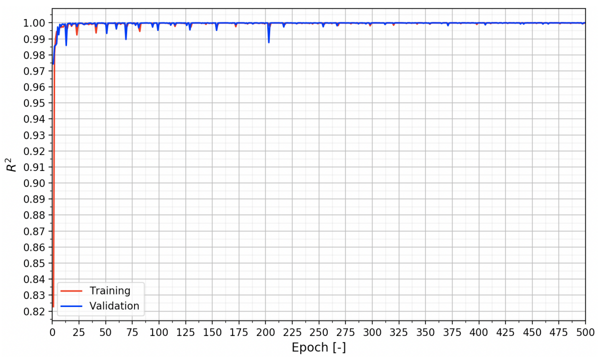


Figure 6.1: R-squared (R^2) evolution over the training process for training and validation datasets.

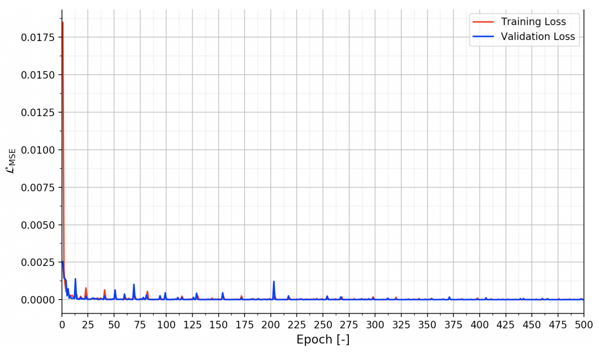


Figure 6.2: Mean-squared-error ($\mathscr{L}_{\text{MSE}}$) evolution over the training process for training and validation datasets.

classification problems than regression-based ones, it is still accounted for by adjusting this monitoring process.

Figures 6.1 and 6.2 show the general performance metrics of the neuromorphic engine. As expected from the hyperparameter analysis, the overall loss, $\mathscr{L}_{\text{MSE}}$ and R^2 metrics are coordinated and saturate very quickly, in less than 20 epochs. However, the training process is forced until 500 epochs to showcase the neuromorphic's engine's ability to maintain the attained level of accuracy. Recall that the R^2 metric is desired to converge as close to 1.0 as possible, whereas $\mathscr{L}_{\text{MSE}}$ should converge to zero. The performance metrics of the two cases (training and validation) support two expected phenomena. The first phenomenon involves the validation error, which is less than the training error in the early stages. Since the validation dataset is held out and only tested after each epoch, it is expected to perform slightly better than the training data which are used through the epoch. The second phenomenon involves the fluctuations in the metrics even while they converge. These fluctuations are expected in the training process of the neural network design process and typically indicate that the batch sizes used are too small. Here again it becomes a delicate balance: choosing batch sizes that are too large, results in a network that converges at a much slower rate. Given that the fluctuations converge to zero with increasing number of epochs, these fluctuations can be disregarded as a whole.

The network converges to an exceptional level of accuracy, despite the complexity of the data repository it is modelling. Table 6.6 summarizes the metrics and the amount of data the network was trained on. Given that all R^2 and MSE values for training, validation, and testing are greater

Table 6.6: Key final performance metrics of the trained neuromorphic engine after 500 epochs.

Parameter	**Symbol**	**Value**
Amount of training datasets	*n.a.*	524,854
Amount of validation datasets	*n.a.*	10,711
Amount of testing datasets	*n.a.*	10,711
R-squared training value	R^2_{training}	0.9999
R-squared validation value	$R^2_{\text{validation}}$	0.9998
R-squared testing value	R^2_{testing}	0.9999
MSE training	$\mathscr{L}_{\text{MSE,training}}$	$4.45 \cdot 10^{-7}$
MSE validatio	$\mathscr{L}_{\text{MSE,validation}}$	$5.32 \cdot 10^{-7}$
MSE testing	$\mathscr{L}_{\text{MSE,testing}}$	$5.04 \cdot 10^{-7}$

than 0.99 and less than 10^{-6} respectively, the neuromorphic engine may be considered sufficiently trained and suitable for deployment. The next step is conducting a cohesive error analysis of the trained network.

6.3.2. Compounded error analysis

Other than the neuromorphic engine's overall performance during the training process, it is important to conduct an error analysis over the entire data repository. Overfitting is the biggest enemy at this stage but cannot be assessed through general performance metrics. Hence, to conduct an error analysis over the entire data repository, the neuromorphic engine's modelling capabilities are compared against all 4,032 models in the data repository. For each geometric configuration, the trained network is evaluated against the FEA results, and the relative error is computed for each data point. Given that all FEA models originate from zero: $f_{e,\text{int}_1}(\Delta a_{e,\text{axial}}) = 0\text{N}$, the error is disregarded for the first data point to prevent division by zero. Furthermore, an error threshold is defined to be 5%: $\varepsilon_{\text{limit}} = 5\%$. With this in mind, an entire error analysis can be conducted encompassing the whole data repository, thereby assessing the neuromorphic engine's modelling capabilities on the provided data.

Two-tone error map: evaluating all tension and compression models

Figure 6.3 shows a post-training error map of all 4,032 specimens of the neuromorphic engine after 10 epochs. Tensile models are shown in blue whereas compression is coloured red. The position of each bubble is dependent on modelling error of the network relative to the FEA deformation curve. The maximum relative error intuitively refers to the largest amount of discrepancy in the dataset of a geometrical configuration. On the other hand, the data fraction above the threshold error is a measure of the amount of data for a simulation which is in excess of the defined threshold

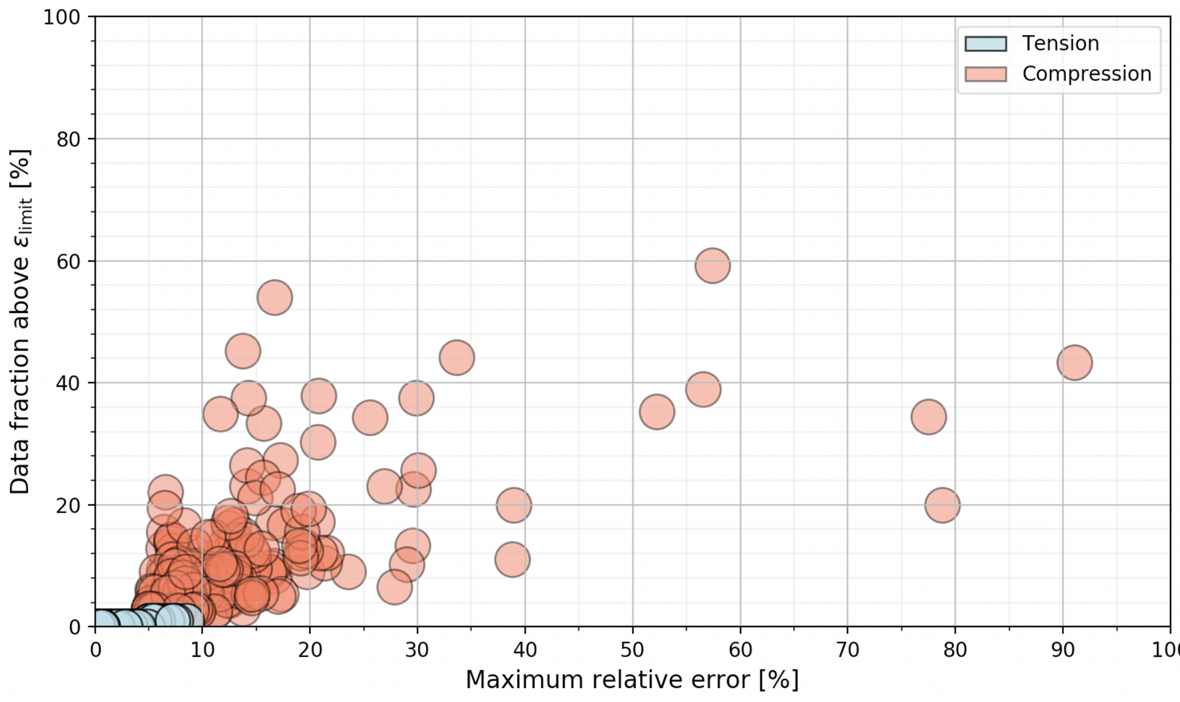


Figure 6.3: Post-training error map between the neuromorphic engine and FEA models across all 4,032 models after 10 epochs.

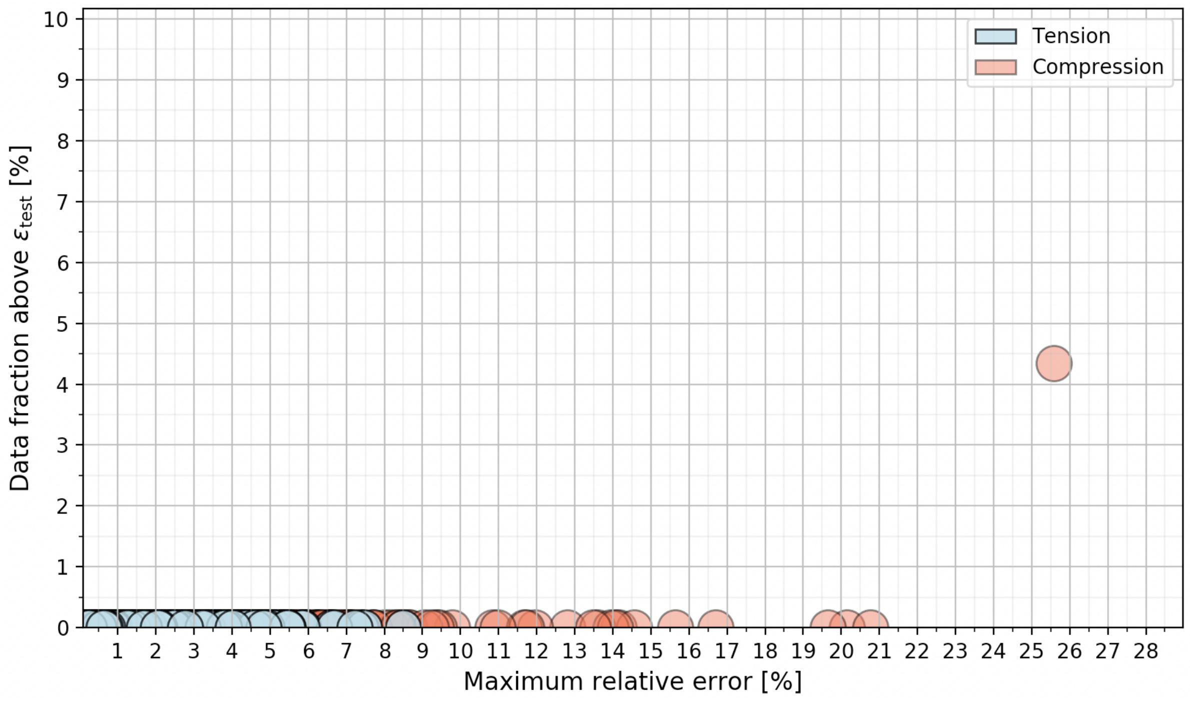


Figure 6.4: Post-training error map between the neuromorphic engine and FEA models across all 4,032 models after 500 epochs.

of 5%. For example, a model with coordinates (20%, 60%) means that the largest error between the FEA model and the neuromorphic approximation is 20% relative to the FEA model; and 60% of the data points are above the 5% threshold. Evidently, this is not acceptable. Ideally, all bubbles should be in the bottom left corner. However, models which have a low data fraction, but a high maximum relative error are also adequate since a single data point with a high error in a near-perfect approximation will not hamper the modelling capabilities of the neuromorphic engine. It is also interesting to note that there are more compression specimens which deviate from the ideal bottom-left-corner than tensile ones. This is expected due to the difficulty in representing the highly nonlinear post-buckling regime.

Figure 6.3 applies to the neuromorphic engine after 10 epochs, but if the training process is extended to 500 epochs, then the results change drastically as may be seen in Figure 6.4. All models converge towards the bottom left corner meaning that the network is capable of modelling in a near-exact fashion every single truss member in the data repository. Although 10 epochs is enough to obtain a good rough approximation, training the neuromorphic engine for 500 epochs is far better and will be used from hereon.

An error map whose axes are both percentages is not a user-friendly visualisation, however it is highly representative of the neuromorphic engine's capabilities. Admittedly, it is also extremely useful for the user to visualise which models are problematic for the network. Unfortunately, Python's graphical GUI does not possess interactive settings to enable easy user-interaction. Because of this, the same error map was produced with an external module known as PlotLy, which builds the same error-map within the computer's internet browser. When hovering the cursor over one of the bubbles in the error map, the geometrical parameters of the considered model and the exact data point in the dataset which corresponds to the maximum relative error. Thereby allowing the user to pinpoint the problematic simulation and conduct a case-by-case evaluation if needed.

Average error across selected data groups

In addition to the error map in Figure 6.3 which shows the network's performance for each individual model, average values are also a good indicator. Furthermore, average values per category can also be highly insightful to the neuromorphic engine's shortcomings. Figure 6.5 summarizes the average errors for some key data groups. As expected, the average error for the tension models is far lower than the compression models due to the severe Nonlinearity in the post-buckling domain. In addition to the tension/compression differentiation, it is also interesting to differentiate between those models that have a radial defect and those having no such defect. It is not entirely surprising those models with a radial defect ($r_{e,D} < r_e$) have a higher average error than those that do not since the presence of a defect induces an additional variation, especially in the compressive case. Although these observations provide interesting insights into the modelling capabilities of the neural network, it should still be noted that the global error is roughly 0.25%, which is more than acceptable from a performance standard.

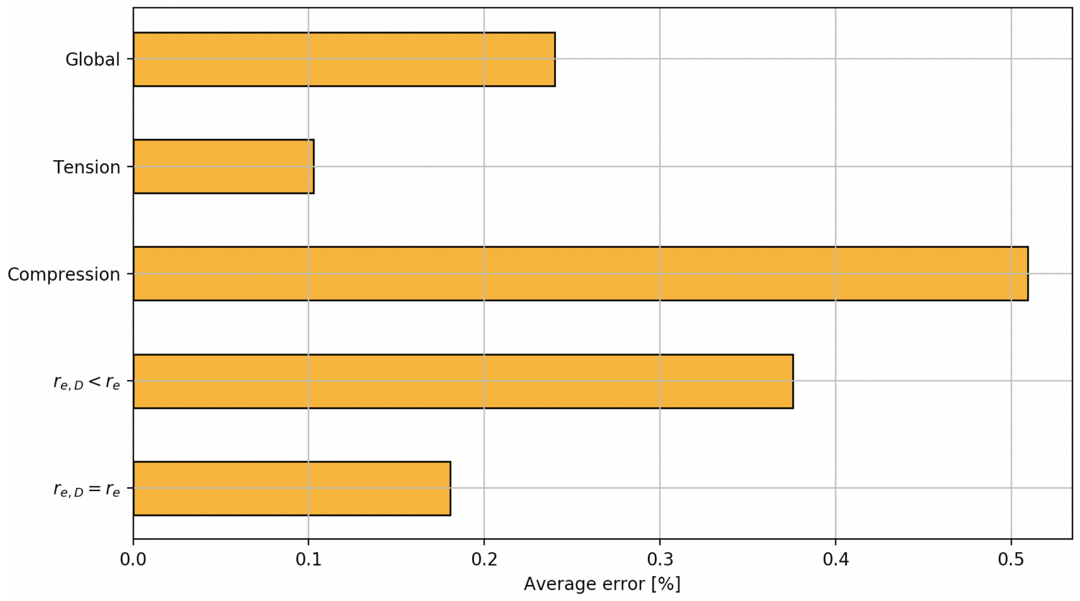
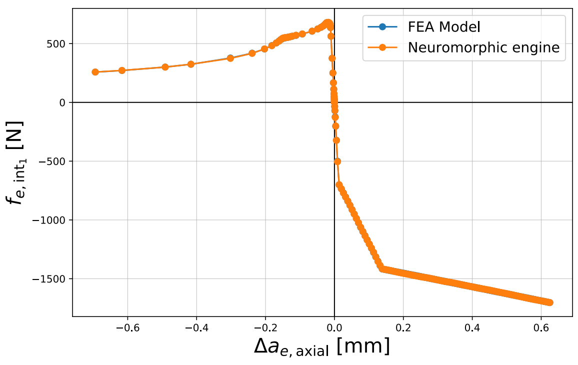


Figure 6.5: Post-training averages errors for selected data groups.

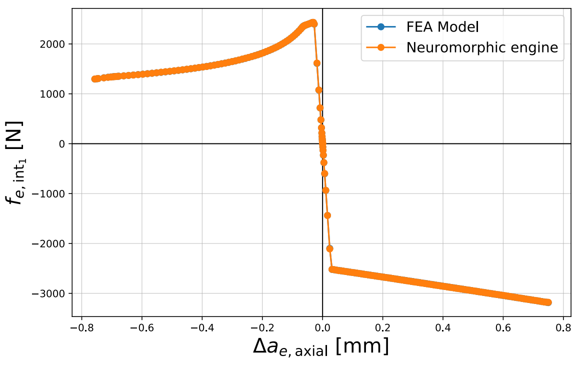
Random model assessment

Random model assessment is one final way the performance of the neuromorphic engine may be investigated. Geometrical parameters are randomly sampled within the parameter design space and the full deformation plot from tensile material plasticity to post-buckling behaviour is constructed. Figure 6.6 compares the modelling capabilities of the neuromorphic engines against 6 randomly selected FEA models from the data repository spanning the entire deformation domain of the specimen from tension to the post-buckling regime, all while including material plasticity. As may be seen, the degree of correlation is almost exact and showcases the expressive capabilities of the trained network. Neither the length, radius and whether there is a radial defect present hampers the accuracy of the network. Despite the complexity and degree of nonlinearity of the entire problem, the neuromorphic engine has no problem capturing the various deformation regimes. In addition, overfitting of any sort is not present.

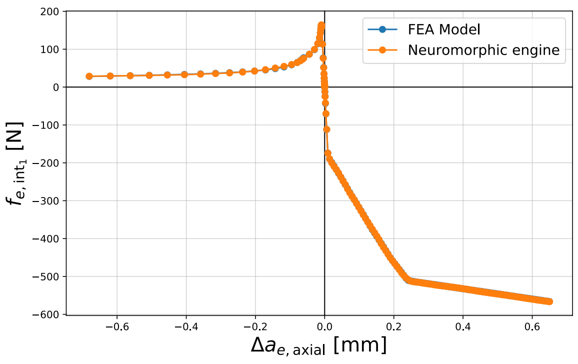
Although assessing the performance of the network against datasets that it was trained and validated against will naturally be accurate, it does not detract from the network's overall modelling capabilities. Given that it is able to model all 2,016 model from tension to post-buckling with a minimal error, the neuromorphic engine can be considered suitable for deployment.



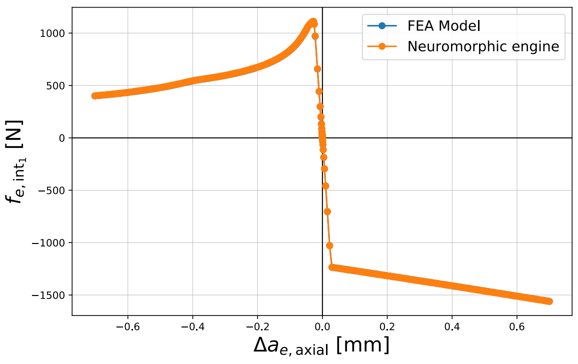
(a) $\{L_e, r_e, r_{e,D}\} = [25.00\text{mm}, 1.50\text{mm}, 1.05\text{mm}]$



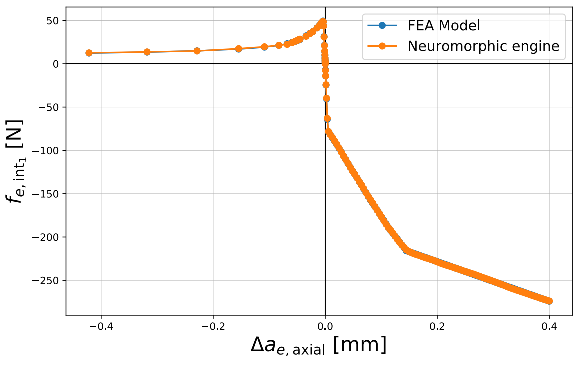
(b) $\{L_e, r_e, r_{e,D}\} = [30.00\text{mm}, 2.00\text{mm}, 2.00\text{mm}]$



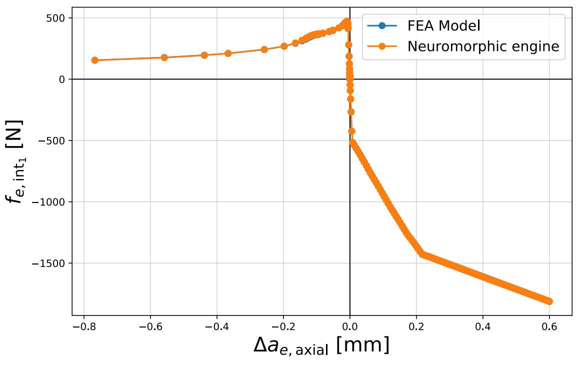
(c) $\{L_e, r_e, r_{e,D}\} = [26.00\text{mm}, 0.90\text{mm}, 0.54\text{mm}]$



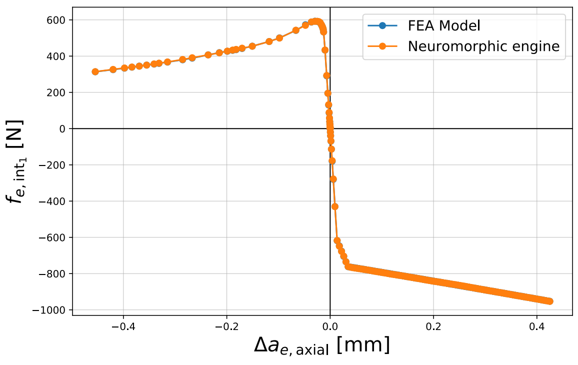
(d) $\{L_e, r_e, r_{e,D}\} = [28.00\text{mm}, 1.40\text{mm}, 1.40\text{mm}]$



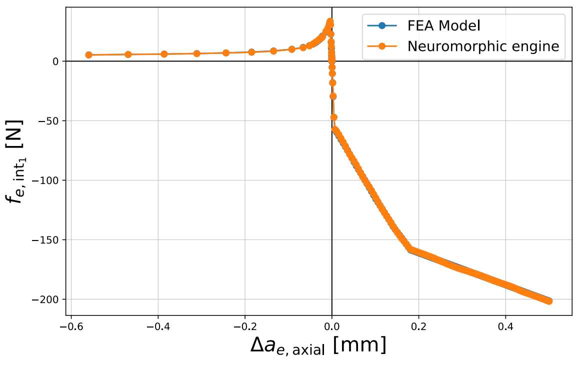
(e) $\{L_e, r_e, r_{e,D}\} = [16.00\text{mm}, 0.70\text{mm}, 0.35\text{mm}]$



(f) $\{L_e, r_e, r_{e,D}\} = [24.00\text{mm}, 1.80\text{mm}, 0.90\text{mm}]$



(g) $\{L_e, r_e, r_{e,D}\} = [17.00\text{mm}, 1.10\text{mm}, 0.99\text{mm}]$



(h) $\{L_e, r_e, r_{e,D}\} = [20.00\text{mm}, 0.60\text{mm}, 0.30\text{mm}]$

Figure 6.6: Modelling capabilities between randomly sampled FEA in the data repository and the trained neuromorphic engine.

6.4. Closure

Building a neural network that chooses its own hyperparameters worked favourably to create the final neuromorphic engine. The global performance measures and error analyses showcase the modelling capabilities of the final product, which is now considered ready for deployment. To summarize, there were two main goals that were achieved in this chapter:

1. Eradicate the trial-and-error approach from the network design, which is known to be extremely time consuming. Volumes could still be written on the sensitivity study of a trial-and-error approach and the effect of the hyperparameters on the network's performance. The trial-and-error approach for neural network as a whole should come to an end since there is no need for them anymore. Data-sampling algorithms which have been present in the fields of computer science for decades can easily be adapted to remove human error from the selection process.

2. Build an engine that is capable of modelling an extremely nonlinear regression problem, on a large-scale parameter design-space.

This serves to underline yet again the potential of machine learning approaches and that their potential is unlimited, provided they are given the proper environment to learn and adequate data. An additional remark concerns the error analysis of the neuromorphic engine, or any trained neural network. From a numerical mathematical standpoint, there is always a constant obsession to reduce the overall error as much as possible. This is not a bad thing, though it is not entirely applicable in the context of machine learning. Stopping the training process of a neural network while there still might be some errors present has been proven to be beneficial, since it helps prevent data-attachment. This refers to the concept that the more a neural network is trained, the more it learns the data that it is trained on. However, the training data are not necessarily representative of the data which it will be deployed on. For instance, in the context of this thesis, the neuromorphic engine might be deployed on a truss member of dimensions $\{L_e, r_e, r_{e,D}\} = [24.82\text{mm}, 1.56\text{mm}, 1.13\text{mm}]$. Although this configuration is within the parameter design space that the model was trained on, there is no model in the data repository which conforms to such values. Therefore, when developing machine learning processes, the obsession of reducing the error into oblivion should be tempered. Rather, error thresholds should be adopted, which are not too restrictive (5% in this case).

Finally, the author firmly believes that the modelling success of the neuromorphic engine is in large part attributable to the fact that the data normalization domain was segregated for $\Delta a_{e,\text{axial}}$ and f_{e,int_1}. By extension, this introduced an additional boundary condition for the network during the training procedure which helped it identify tensile and compressive cases. Should this additional data segregation step have been overlooked, and all the data had been normalized to a standard $[0,1]$ domain, then the neuromorphic engine's modelling capabilities would not have been as accurate as they were, and a far more complex neural architecture would have been required to attain the same level of expressiveness.

At this stage, the neuromorphic engine will no longer be re-trained, analyzed, or disturbed in any fashion. It is now considered as what engineers refer to as a *black box*. It is saved as a singular HDF5 file (see Chapter 7), which contains all the trained synaptic weights, biases, and other hyperparameters; it can be opened in either Python or Matlab. The next steps concern the wiring of this black box into a user-element for finite element analyses, which will be described in the following chapter.

7 NmT2: A New Class of Element

The aim of this thesis is to produce a new element that can readily be implemented in an active FEA simulation to improve computational efficiency without compromising accuracy. This is achieved through the use of a neuromorphic engine trained on a properly built dataset to model axial deformation of a truss-member. The previous chapter focused on building and honing the neuromorphic engine, which constitutes a significant portion of the finite element development. But the custom element is not yet suitable for deployment inside an active FEA simulation. The neuromorphic engine has to be appropriately formatted in terms of a user-subroutine for usage in ABAQUS. The final product represents a new form of finite element. It also introduces an entire new class of elements. This new class of elements intelligent machine learning to reduce computational expense while providing high-fidelity analyses of multi-truss structures. Additional metrics include the ability to model material plasticity and the post-buckling regime. This new class of element is from hereon referred to as a *neuromorphic element*. For the specific setting of this project, which focuses on building an element in 2D space, the following element reference code is developed:

$$\begin{tikzcd}
 & \text{Truss Member} & \\
\text{Nm} \arrow[d] & \text{T} \arrow[u] & 2 \arrow[d] \\
\text{Neuromorphic Class} & & \text{2D Space}
\end{tikzcd}$$

The presence of the *Truss Member* and *2D Space* descriptors implies that the neuromorphic element class can be extended to 3D space and be used to model other structural components (plates and shells for example). This is further discussed at the end of this thesis when contemplating future work (Chapter 12). The current chapter focuses on the process of building the neuromorphic engine into the neuromorphic NmT2 element. Although this is a relatively simple procedure which mostly involves understanding and translating between various data formats, it merits its own chapter to underline the deployment procedure.

7.1. Python - FORTRAN translator

It is important to understand that the trained neuromorphic engine cannot be directly exported within an ABAQUS user-element. ABAQUS reads user subroutines in FORTRAN language whereas the neuromorphic engine was developed in Python. Therefore, a translator is required to translate the neural network from Python into FORTRAN.

The first step is saving the trained network in a suitable format. Hierarchical data formats, especially HDF5, are typically used to export and save machine-learning programs built in Python.

HDF5 files essentially store an entire directory within a single file and are thereby perfectly suited for large and complex programs with their own unique architecture. Neural networks are a prime example of programs that have their own unique architecture and benefit from such file formatting. A neural network can be exported and replicated exactly if the following are known:

- Input layer format (number of inputs).
- Output layer format (number of outputs).
- Number of hidden layers.
- Number of nodes in each hidden layers (≈ network shape).
- Activation function used in hidden layers.
- Activation function used in output layer.
- Values of all synaptic weights.
- Values of all nodal biases (if used).

The above parameters are all that are required to replicate a trained neural network exactly. The scaling of the input data is also of crucial importance, although this is not included when formatting the network. By saving the parameters in a hierarchical data structure, the trained network can be exported and used by anyone without the need of a training database, sensitivity analysis, or optimization scheme.

Unlike Python, FORTRAN is a compiled language and does not benefit from interpretive coding. This means that to pass a neural network through FORTRAN code, it has to be reduced to sequential operations. As outlined in Chapter 2, for a given input vector $\{\mathbf{x}\}$, the output of the first network layer, $\mathbf{o}_i$, can be computed as:

$$\mathbf{o}_i = \psi_i\Big(\mathbf{W}_i^{\mathrm{T}}\{\mathbf{x}\} + \mathbf{b}_i\Big) \qquad \text{for: } i = 1 \tag{7.1}$$

Where ψ_i, $\mathbf{W}_i^{\mathrm{T}}$ and $\{b\}_i$ are, respectively, the activation function, trained weights, and biases of each layer. For the subsequent layers, the outputs of each layer are:

$$\mathbf{o}_i = \psi_i\Big(\mathbf{W}_i^{\mathrm{T}}\mathbf{o}_{i-1} + \mathbf{b}_i\Big) \qquad \text{for: } i > 1 \tag{7.2}$$

In essence, all that is required to build the network are the activation functions used, and the trained weights and biases. The dimensions of the layers in the network can be inferred from the vector size of either the synaptic weight or bias vectors. Given that this information is stored in the HDF5 file, it is possible to re-write the network such that FORTRAN can interpret it. This program was developed such that it is not specific to the context of this thesis, and therefore may be used to translate any neural network saved in HDF5 format into FORTRAN code. From here, the developed neuromorphic engine can be built into FORTRAN, which is the first step in finalizing the UEL.

7.2. NmT2 Deployment

The framework of the neuromorphic element and how it specifically interacts with the ABAQUS/Standard solver was described in Chapter 4 and will not be reiterated here. The entire UEL of the NmT2 element could be written as a standalone subroutine. But this approach is not preferable given that the length of the FORTRAN code can become very long. For example, a neural network with four hidden layers and 25 nodes in each layer produces 455 lines of FORTRAN code, thereby increasing the risk of programmer error within the UEL. Therefore, the preferred approach is to build the neuromorphic engine into the UEL via a separate module that can be called when needed. The interaction process between the ABAQUS input file and the neuromorphic element may be seen in Figure 7.1.

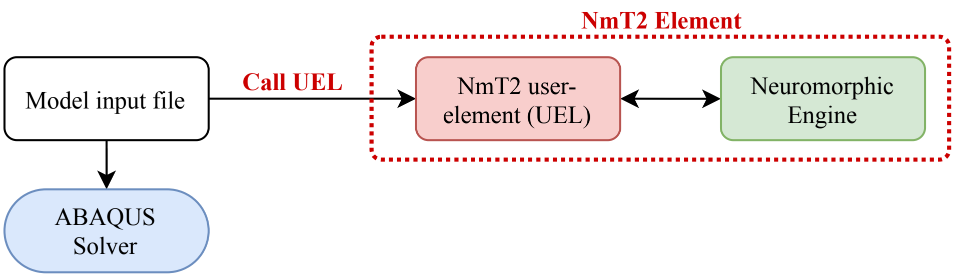


Figure 7.1: Interaction between the model input file and neuromorphic element NmT2 prior to being submitted to the ABAQUS solver.

Note the double-headed arrow between the UEL and neuromorphic engine. The role of the NmT2 user-element is to provide the ABAQUS solver with the required outputs. To compute the nodal forces, the traditional methods are replaced by invoking the neuromorphic engine, which then feeds the results back into the NmT2 UEL. As mentioned before, these two components could potentially be merged into one, though this approach permits a more elegant format for the UEL and minimizes the introduction of human error during development.

The last step involves calling the NmT2 element through the input file, which is conducted in the same manner as any user-subroutine. With the neuromorphic engine translated into FORTRAN and successfully interacting with the NmT2 UEL, the element is ready to be deployed and tested on a series of case studies.

8

Case Study 1: Single Truss Member

The first logical case study is that of a single horizontal truss member, loaded axially. This corresponds to the same kind of model that was used to build the data repository. Such a verification also provides a first step and a benchmark for potential applications to multi-truss structures. Therefore, just as before, consider a horizontal truss member that is pinned at the left end and simply supported at the right-most end as shown in Figure 8.1. The beam is displacement-loaded at its right end and the reaction force at the opposite end is extracted as f_{e,int_1}. The geometrical parameter design space includes a structural defect which ranges within $r_{e,D} \in [0.5r_e, r_e]$. As described before, in the case of tension, the structure can be meshed with only 3 T2D2 elements (as shown in Figure 8.1). Compression, however, may contain several beam elements including mesh bias de-

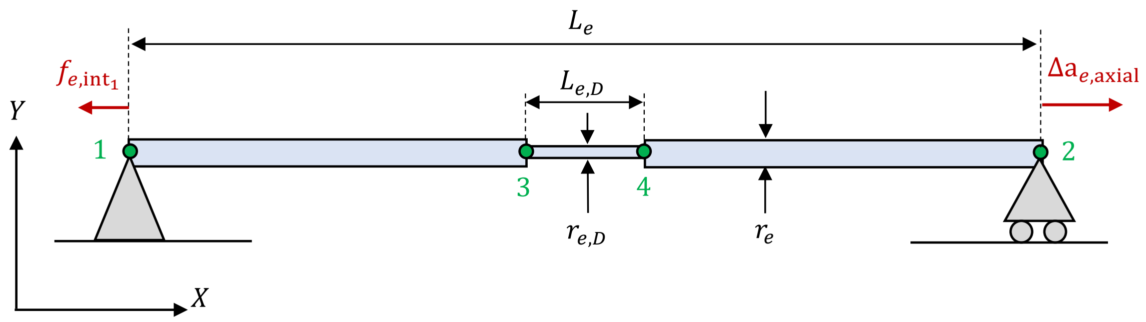


Figure 8.1: Case study 1: simply-supported beam with displacement loading (in this case, the structure is loaded in tension).

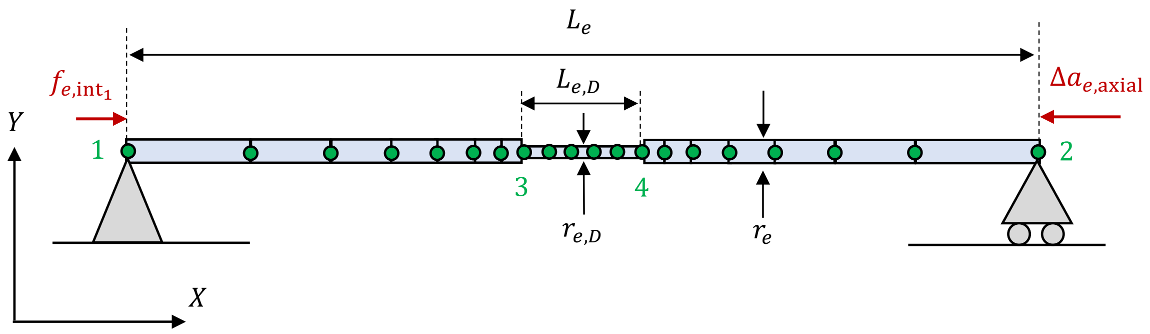


Figure 8.2: Case study 1: simply-supported beam with displacement loading (in this case, the structure is loaded in compression, note the use of extra nodes to model accurately the post-buckling deformation).

pending on the lengths used (an example is shown in Figure 8.2). If the neural element is used, then the entire structure can be replaced with a single NmT2 element; there is no need for meshing the structure (Figure 8.3). Rather, the neuromorphic element possesses the same geometrical properties as the meshed structure, which is theoretically sufficient to replicate the deformation field both in the case of tension and compression, all while including material plasticity.

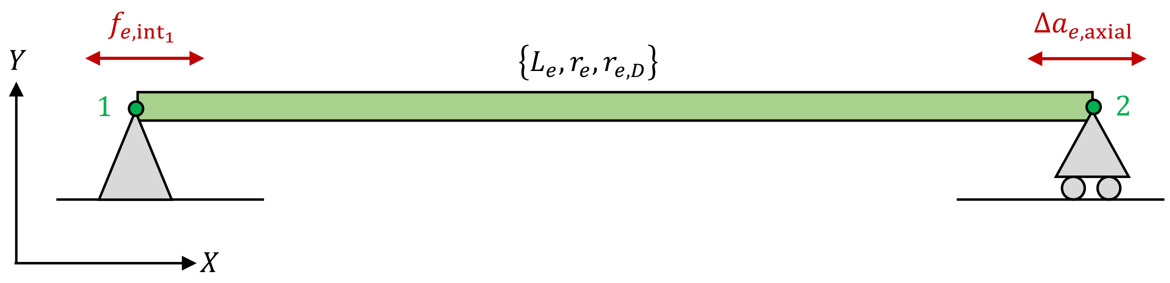


Figure 8.3: Case study 1: simply-supported beam with displacement loading using a single NmT2 element that has the same geometry as the truss member.

Given that the NmT2 element is purely data-driven, it is expected to be far more computationally efficient than a traditional FEA analysis. This section evaluates the computational gains (if any) and accuracy between a traditional FEA analysis and one using the newly developed NmT2 element for a simply-supported structural member.

8.1. Preparing the analysis

The easiest way to submit a cohesive structure for analysis through ABAQUS is via an input file. It is important to recall that once the geometrical parameters are established, the entire loading space of the beam is defined by extension (since $\Delta a_{e,i+1_{\text{axial}}} \in [0, 0.5L_{e,D}]$). Therefore, once the beam length, radius, and defect are chosen (L_e, r_e, and $r_{e,D}$ respectively), the geometry and loading domain of the structure are defined. The only additional parameter that the user must select is whether the structure is loaded under tension or compression, which in turn affects the sign of $\Delta a_{e,i+1_{\text{axial}}}$.

To prepare the analysis, a Python program is devised which builds two input files depending on the global geometrical parameters. One input file uses the traditional FEA elements; the other uses NmT2 elements. In the case of the traditional FEA analysis, a few extra parameters are built in for the user to choose the mesh density, loading step size, and element type. These do not affect the analysis using NmT2 elements. It is also important to note that for the structure using the NmT2 element, the input file must reference the neuromorphic element and neuromorphic engine sub-

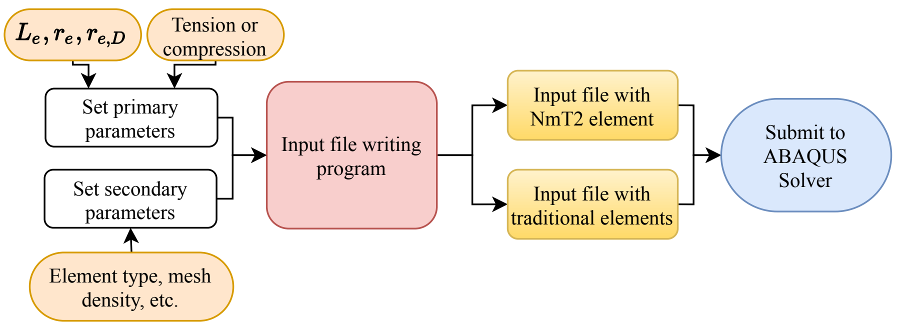

$L_e, r_e, r_{e,D}$
Tension or compression
Set primary parameters
Set secondary parameters
Element type, mesh density, etc.
Input file writing program
Input file with NmT2 element
Input file with traditional elements
Submit to ABAQUS Solver

Figure 8.4: Flow process to prepare the FEA analyses. One using traditional elements, and the other using a single NmT2 element.

routines as described in the previous chapter. The entire flow process for the ABAQUS analysis is summarized by Figure 8.4, which will also be used in the following case studies.

8.2. Results

In order to prevent user-bias, the primary geometrical parameters are randomly sampled within the prescribed parameter design space defined in Chapter 4. A total of 30 random configurations are produced to test the capabilities of the neural element for both the tensile and compressive cases. This means that no identical model is subjected to both tension and compression. Therefore, in total, 60 models are tested. For clarity, the results are divided between the tensile and compressive cases; they are merged together at the end of the analysis. To reiterate, the purpose of these case studies is to assess the capabilities of the NmT2 element both in terms of accuracy and computational efficiency relative to a traditional FEA model.

8.2.1. Tension

In the case of a tensile FEA model, 3 elements are sufficient as shown before and are initially set to classic T2D2 elements. The main metric used to measure the accuracy between the traditional FEA models and the neuromorphic element is simply the absolute errors: the lowest, highest, and average percentage errors (denoted as ϵ_{min}, ϵ_{max} and $\bar{\epsilon}$ respectively) are computed for each simulation relative to the traditional FEA models. However, it should be emphasized that there is no longer an exact data match between the FEA and NmT2 models: unlike when analyzing the performance of the neuromorphic engine relative to the data repository (meaning that $(\Delta a_{e,\text{axial}})^{(i)}_{\text{NmT2}} \neq (\Delta a_{e,\text{axial}})^{(i)}_{\text{T2D2}}$. In this case the errors are extrapolated between the two closest data points. In addition to the absolute differences, the data imbalance lends itself well to the R^2 metric which was introduced in Chapter 6 when building the neuromorphic engine. The reason the R^2 value is preferred as an error evaluation over other conventional statistical metrics (such as the root mean-squared error, L2 norm, etc.) is that it provides a dimensionless quantity that represents the degree to which an approximation regression fits a reference model. Recall that R^2 can take on any real value from zero to unity. The closer it is to 1.0, the better the statistical fit. Additionally, the computational processing time is also reported: t_{T2D2} for the models meshed with T2D2 elements and t_{NmT2} for those with a singular neuromorphic element.

Table 8.1 shows all the results for the tensile case and the geometrical models that were randomly generated as well as the overall average values of the absolute error metrics, R^2 value, and the computational time required for a simulation to complete. At this stage, only an error analysis is conducted. The graphical comparisons are left for the combined comparison between the compressive and tensile cases.

In the tensile case, the average absolute error is roughly 1.0% which showcases the modelling capabilities of the NmT2 element relative to traditional T2D2 elements. Although the absolute error metrics are useful, the most important metric is the R^2 value, which is indicative of the overall approximation of the regression. As may be seen in Table 8.1, $R^2 \approx 1$ across all 30 models indicating a near-exact approximation. The R^2 metric is naturally lower than when using it to evaluate the neuromorphic engine, since it is now being deployed on independent cases as opposed to the cases sampled from the pre-existing data repository. Despite this, the performance of the neuromorphic element in a live FEA setting is exceptional and is impervious to the presence of a defect or other geometrical parameters. Furthermore, regarding computational performance, the NmT2 element reduces the computational time by almost half. The T2D2 elements average roughly 40 seconds per FEA simulation whereas those with neuromorphic elements average 24 seconds. Although this might seem like a small difference, it can become substantial if it holds when the complexity of the FEA model is scaled up, as will be seen in the following section and chapters.

Before proceeding to the compression case, it is worth addressing outliers identifiable through the maximum absolute error, $\epsilon_{\max}$. In some circumstances, the maximum $\epsilon_{\max}$ peaks at 20%. If this difference is located at a key design point in the load-bearing capability of the structure, then it might pose a problem. However, given that $\bar{\epsilon}_{\max} \approx 7\%$ over 30 randomly generated models, it remains relatively low risk. From an error analysis standpoint, the results from the tensile case are more than sufficient to verify the capabilities of the NmT2 element. From here, a similar error analysis may be conducted for the compression case.

Table 8.1: Error and temporal analysis for all tensile cases between traditionally meshed models using T2D2 elements and models using a single NmT2 element.

L_e [mm]	r_e [mm]	$r_{e,D}$ [mm]	$\epsilon_{\max}$ [%]	$\epsilon_{\min}$ [%]	$\bar{\epsilon}$ [%]	R^2 [-]	t_{T2D2} [s]	t_{NmT2} [s]
13.85	1.44	0.76	3.901	0.156	2.271	0.994	39.75	25.56
14.05	0.65	0.40	6.090	0.004	1.496	0.994	42.01	26.78
14.28	0.88	0.59	2.509	0.128	0.878	0.994	38.98	25.76
16.23	1.23	1.16	15.134	0.013	1.137	0.993	37.65	24.12
17.08	1.08	0.98	2.174	0.010	0.654	0.994	36.99	25.55
17.51	1.31	0.78	4.815	0.025	0.799	0.995	38.76	26.01
17.95	1.78	1.73	3.780	0.013	0.407	0.997	42.70	25.89
18.45	0.76	0.39	7.010	0.108	2.194	0.997	39.45	22.23
18.87	1.45	0.83	5.001	0.002	1.868	0.996	39.12	24.66
18.94	0.76	0.40	9.034	0.133	2.473	0.996	38.57	24.56
20.05	1.49	1.11	4.939	0.015	0.618	0.997	38.85	25.06
20.14	1.79	1.63	3.323	0.004	0.529	0.997	44.17	24.13
20.63	1.39	0.95	3.864	0.041	0.572	0.997	40.72	25.29
20.70	0.57	0.36	7.160	0.017	1.532	0.997	39.02	25.64
20.97	1.42	1.15	2.345	0.002	0.345	0.998	39.25	23.76
21.86	1.29	1.18	7.416	0.004	0.563	0.998	43.27	21.36
22.29	1.00	0.75	2.278	0.015	0.779	0.998	42.12	24.04
22.86	1.05	0.89	4.372	0.036	0.624	0.998	40.34	25.99
23.10	1.14	0.75	2.304	0.087	0.948	0.998	35.79	26.11
23.32	0.62	0.59	14.267	0.003	1.380	0.998	34.51	24.36
23.93	0.98	0.49	7.014	0.087	0.969	0.998	39.55	24.99
24.66	0.64	0.54	10.877	0.003	1.656	0.998	39.01	23.03
24.72	1.29	1.28	14.34	0.000	1.015	0.997	40.19	25.24
24.89	1.59	1.35	3.307	0.006	0.509	0.997	45.29	23.44
25.19	0.86	0.43	3.646	0.028	0.991	0.997	39.77	25.57
25.39	1.95	1.92	20.367	0.014	1.375	0.994	34.24	23.13
25.49	0.59	0.35	4.749	0.049	1.426	0.994	42.13	21.98
26.06	1.13	1.09	11.80	0.010	1.032	0.994	39.66	24.06
26.69	1.76	1.66	16.36	0.029	1.039	0.994	39.98	23.46
28.30	1.36	1.07	2.682	0.033	0.519	0.994	38.65	22.53
Average			6.895	0.036	1.087	0.996	39.68	24.48

8.2.2. Compression

The compression error analysis is carried out in the exact same manner as the tensile analysis, across the same 30 models. However, the models are meshed with B22 elements in the case of the traditional FEA analysis to allow for deformation out of the axial plane. The number of elements and mesh convergence will not be discussed here as to not detract from the focus of this chapter. However, all convergence was checked for each case following the same method as outlined in the data-generation process of Chapter 5.

Table 8.2 shows the error analysis conducted across 30 models loaded in compression and into the post-buckling regime, all while including material plasticity. Relative to the tensile model, all the errors have increased, which is expected due to the nonlinear nature of the post-buckling regime.

Table 8.2: Error and temporal analysis for all compressive cases between traditionally meshed models using B22 elements and models using a single NmT2 element.

L_e [mm]	r_e [mm]	$r_{e,D}$ [mm]	$\epsilon_{\max}$ [%]	$\epsilon_{\min}$ [%]	$\bar{\epsilon}$ [%]	R^2 [-]	t_{B22} [s]	t_{NmT2} [s]
13.85	1.44	0.76	20.802	0.279	6.446	0.919	89.31	22.01
14.05	0.65	0.40	19.449	0.330	6.325	0.919	79.19	24.24
14.28	0.88	0.59	22.217	0.506	5.243	0.917	86.48	23.59
16.23	1.23	1.16	13.257	0.407	5.839	0.923	85.93	19.37
17.08	1.08	0.98	13.634	0.161	5.974	0.921	88.17	21.28
17.51	1.31	0.78	18.793	0.329	7.134	0.922	81.38	20.01
17.95	1.78	1.73	11.224	0.859	5.453	0.934	83.05	19.92
18.45	0.76	0.39	32.364	0.899	12.004	0.935	85.12	23.93
18.87	1.45	0.83	20.887	0.086	8.221	0.933	84.09	26.38
18.94	0.76	0.40	39.989	2.187	15.540	0.933	80.38	23.08
20.05	1.49	1.11	20.737	0.314	6.498	0.939	88.19	25.37
20.14	1.79	1.63	4.226	0.189	1.796	0.939	90.01	24.18
20.63	1.39	0.95	17.941	0.0367	4.696	0.958	89.27	18.38
20.70	0.57	0.36	10.336	0.857	13.885	0.959	85.88	18.29
20.97	1.42	1.15	9.445	0.184	3.219	0.962	83.36	20.20
21.86	1.29	1.18	9.85	0.373	3.075	0.964	79.98	21.39
22.29	1.00	0.75	17.619	0.261	4.644	0.964	91.05	23.39
22.86	1.05	0.89	10.001	0.157	4.331	0.965	88.38	22.38
23.10	1.14	0.75	7.966	0.517	2.790	0.965	88.42	24.49
23.32	0.62	0.59	23.256	0.248	5.964	0.965	82.34	20.29
23.93	0.98	0.49	17.545	0.450	5.022	0.965	86.96	20.98
24.66	0.64	0.54	15.413	0.116	6.404	0.965	81.19	18.28
24.72	1.29	1.28	7.910	0.259	3.093	0.967	85.38	17.89
24.89	1.59	1.35	6.903	0.187	3.122	0.969	88.87	24.10
25.19	0.86	0.43	11.588	0.178	4.407	0.969	87.79	25.01
25.39	1.95	1.92	3.854	0.121	1.667	0.978	81.08	24.98
25.49	0.59	0.35	37.347	1.548	19.329	0.978	88.00	24.67
26.06	1.13	1.09	9.164	0.227	3.212	0.978	83.01	20.38
26.69	1.76	1.66	5.036	0.045	1.514	0.981	80.89	28.97
28.30	1.36	1.07	12.817	0.300	4.582	0.981	90.83	19.89
Average			15.719	0.420	6.048	0.952	85.47	22.24

However, the R^2 value, although not as high as in the tensile case, still remains above 0.9 indicating a high degree of correlation with the FEA model (over 30 simulations, the average $R^2 = 0.952$). A correlation is observable between the length of the specimen L_e and R^2 indicating that the neuromorphic element has a greater difficulty modelling short and stout truss members as opposed to longer ones. Nevertheless, the difference remains small. Furthermore, loading a structure past the point of buckling drastically increases the computational effort which now averages 85 seconds for models meshed with B22 elements. However, the neuromorphic elements retain the same computational efficiency averaging around 22 seconds.

The absolute errors have all increased relative to the tensile case reaching a maximum of 40% deviation in one case. The average error, $\bar{\epsilon}$ is around 6% which is a six-fold increase when compared to the tensile case. Overall, the error analysis is still satisfactory, mostly due to the R^2 values which are still concentrated above 0.9. Regarding the computational efficiency however, data-driven processes start to excel relative to traditional FEA processes, resulting in nearly a 300% increase in computational efficiency. Whether the increase in computational efficiency is sufficient to justify the slight loss in accuracy remains debatable and will be reviewed at the end of this chapter. For now, the results in compression are just as promising as those from the tensile case, despite the slightly larger error gap in the compression analysis.

8.2.3. Global analysis

As in previous chapters, the tensile and compressive deformation regimes can be merged to form the entire structural response of the single horizontal truss member. The same error analysis conducted separately on the tensile and compressive cases can also be done for the combined case, which is shown in Table 8.3. However, in this combined case, the maximum and minimum errors (ϵ_{max} and ϵ_{min}) are omitted since they would intuitively refer to the same values for either the tensile or compressive case. Additionally, the computational time is omitted: it is not representative of an actual FEA simulation since this global analysis is the result of two merged analyses. Only the average absolute error and R^2 metric are reported and are considered sufficient to assess the difference between traditional elements and a singular NmT2 element.

Regarding the R^2 metric, as described during the development of the neuromorphic engine in Chapter 6, it is unwise to apply blindly the error descriptor to the entire deformation curve which has an output range that spans from negative to positive. This forces the mean of the overall deformation plot closer to zero, which in turns leads to a higher R^2 value that is not representative. Rather, the problem should be treated as two independent regression problems that are assembled together. Hence the R^2 metric of the global analysis is simply the average between the tensile and compression cases. With this in mind, the R^2 values reported in Table 8.3 are well above 0.9 indicating that the NmT2 element provides an acceptable approximation to the model using traditional FEA elements. On the other hand, the average absolute error is not as indicative of the quality of the approximation as the R^2 metric, however it remains useful as an indicator. The global average absolute error, being mostly influenced by the compression case, is just over 3%, which is acceptable for verification purposes.

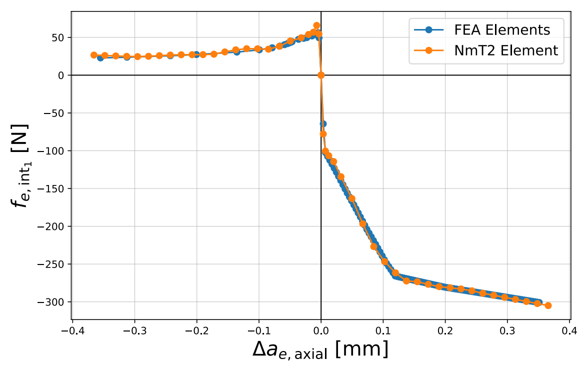
The full deformation cycles of 8 randomly generated specimens are shown in Figure 8.5. Eight were selected which best captured both the variation of geometrical configurations as well as the various modelling problems the NmT2 has relative to traditional FEA elements.

The first observations concern the tensile part of the curve which is the portion on the right (where $\Delta a_{e,\text{axial}} > 0$. The neuromorphic element shows one type of abnormal response which is a slight oscillatory behaviour nearing the end of the tensile deformation regime. This is slightly visible in Figure 8.5a and is more prominent in Figures 8.5b and 8.5c. Overfitting would be the typical culprit when such behaviour is present. However, the oscillatory behaviour fades when the length of specimen is increased, and no overfitting was present during the testing phases of the neuromorphic engine. This behaviour is only observed in smaller specimens where $L_e < 19.0$mm. It will be kept in mind during the evaluation of the other case studies before forming a final conclusion.

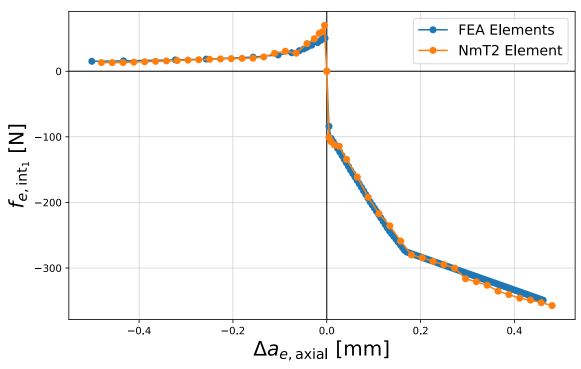
On the compression side where $\Delta a_{e,\text{axial}} < 0$, the error discrepancies become more visible. The NmT2 element tends to overshoot the buckling peak which is most visible in Figures 8.5a through 8.5c. This is arguably the most difficult part of the post-buckling curve for the element to model; the maximum absolute error consistently stems from this part of the curve. The remainder of the deformation curve is quite well approximated by the neuromorphic element and always converges to the same value as in the case of traditional FEA elements. The only other problem that occurs is the relaxation rate in the post-buckling domain which is often overestimated by the NmT2 element and most visible in Figure 8.5d. These two areas represent the height of nonlinearity in the overall deformation regime and are the most difficult to model. For the remainder, the compression deformation regime is almost perfectly modelled and any discrepancies in the curve disappear with increased specimen length L_e leading to a near-perfect deformation plot.

Table 8.3: Error analysis for all cases between traditionally meshed models using standard FEA elements (either B22 or T2D2) and models using a single NmT2 element.

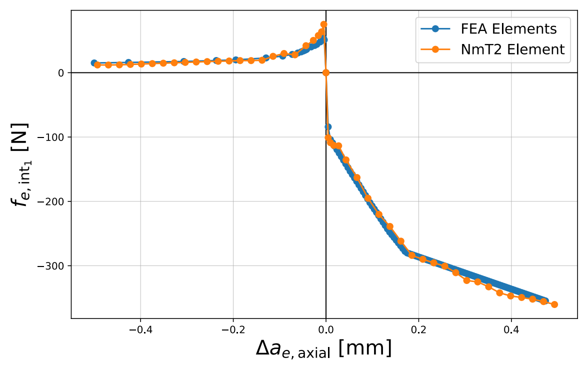
L_e [mm]	r_e [mm]	$r_{e,D}$ [mm]	$\bar{\epsilon}$ [%]	R^2 [-]
13.85	1.44	0.76	4.452	0.956
14.05	0.65	0.40	3.962	0.957
14.28	0.88	0.59	4.237	0.955
16.23	1.23	1.16	3.538	0.958
17.08	1.08	0.98	3.371	0.957
17.51	1.31	0.78	4.034	0.956
17.95	1.78	1.73	2.984	0.956
18.45	0.76	0.39	7.203	0.966
18.87	1.45	0.83	5.112	0.965
18.94	0.76	0.40	9.146	0.954
20.05	1.49	1.11	3.621	0.967
20.14	1.79	1.63	1.176	0.978
20.63	1.39	0.95	2.678	0.978
20.70	0.57	0.36	7.840	0.968
20.97	1.42	1.15	1.813	0.979
21.86	1.29	1.18	1.846	0.981
22.29	1.00	0.75	2.778	0.981
22.86	1.05	0.89	2.517	0.981
23.10	1.14	0.75	1.889	0.981
23.32	0.62	0.59	3.725	0.981
23.93	0.98	0.49	3.039	0.981
24.66	0.64	0.54	4.080	0.981
24.72	1.29	1.28	2.076	0.982
24.89	1.59	1.35	1.843	0.983
25.19	0.86	0.43	2.735	0.983
25.39	1.95	1.92	1.524	0.986
25.49	0.59	0.35	10.568	0.986
26.06	1.13	1.09	2.145	0.986
26.69	1.76	1.66	1.281	0.987
28.30	1.36	1.07	2.594	0.987
Average			3.660	0.973



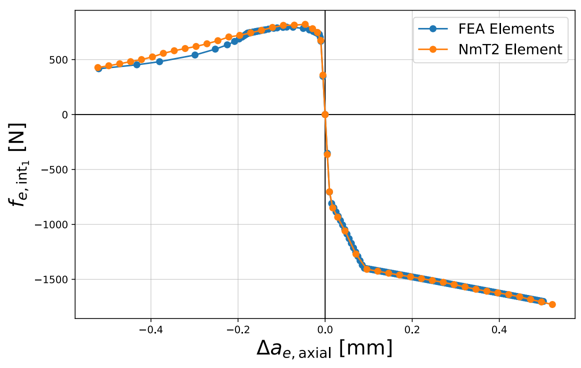
(a) $\{L_e, r_e, r_{e,D}\} = [14.05\text{mm}, 0.65\text{mm}, 0.40\text{mm}]$



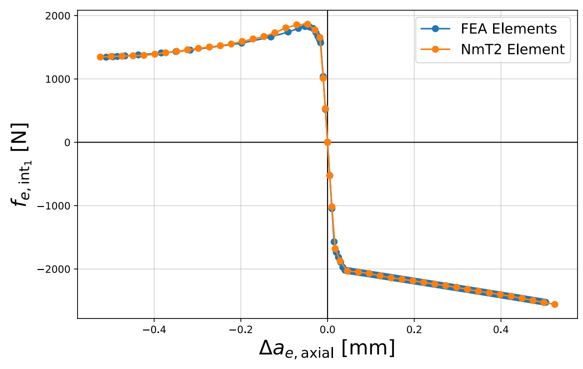
(b) $\{L_e, r_e, r_{e,D}\} = [18.45\text{mm}, 0.76\text{mm}, 0.39\text{mm}]$



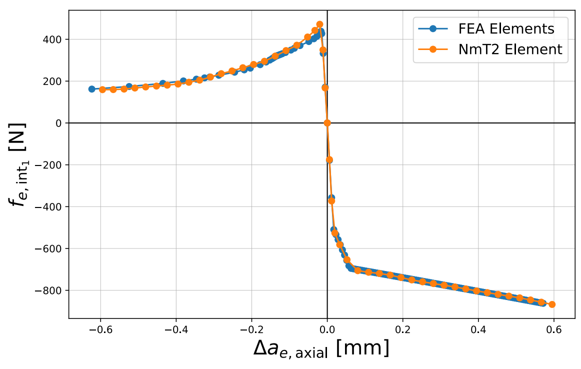
(c) $\{L_e, r_e, r_{e,D}\} = [18.94\text{mm}, 0.76\text{mm}, 0.40\text{mm}]$



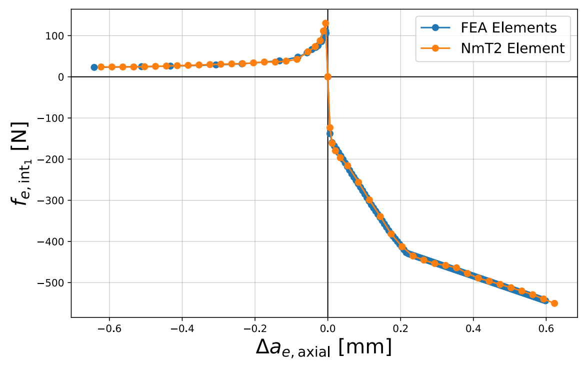
(d) $\{L_e, r_e, r_{e,D}\} = [20.05\text{mm}, 1.49\text{mm}, 1.11\text{mm}]$



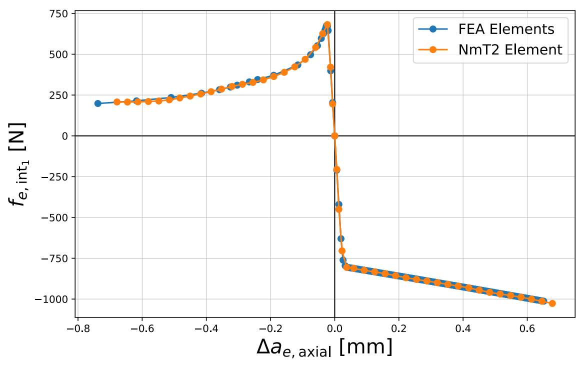
(e) $\{L_e, r_e, r_{e,D}\} = [20.14\text{mm}, 1.79\text{mm}, 1.63\text{mm}]$



(f) $\{L_e, r_e, r_{e,D}\} = [22.86\text{mm}, 1.05\text{mm}, 0.89\text{mm}]$



(g) $\{L_e, r_e, r_{e,D}\} = [23.93\text{mm}, 0.98\text{mm}, 0.49\text{mm}]$



(h) $\{L_e, r_e, r_{e,D}\} = [26.06\text{mm}, 1.13\text{mm}, 1.09\text{mm}]$

Figure 8.5: Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (either T2D2 or B22 shown in blue), and the same models meshed with a single NmT2 element (shown in orange).

8.3. Closure

The purpose of this case study was to start assessing the capabilities of the neuromorphic element in a simplified setting, which was identical to that defined during the data-generation process. Overall, the NmT2 element performed very well when compared to traditional FEA elements such as T2D2 or B22. For tensile loading, the results were near-exact relative to the standard FEA deformation profiles. The severe kinks due to material plasticity posed no problem whatsoever. When loaded in compression, the correlation with the FEA model was just as good, with a few extra discrepancies which are expected due to the nonlinear nature of the post-buckling regime. The sources of the discrepancies stemmed mostly from models where the length of the truss member was less than 20mm, which will be kept in mind while moving forward.

Throughout the entire process, the computational time of the NmT2 element remained much lower than that of B22 or T2D2 elements. This increase in computational efficiency of almost 300% in the compressive case already indicates that the neuromorphic element shows significant potential for reducing computing time. Although the trade-off between solution accuracy and computational effort will be a topic for Chapter 11, at this stage the results are very promising.

Another important aspect is that the number of iterations required by the ABAQUS solver is far less for the NmT2 element than for traditional FEA elements. This difference in computational efficiency is even more pronounced for the compression case when modelling the post-buckling regime. Overall, across the entire deformation curve for a given model, the NmT2 elements require 75% fewer increments than those meshed with traditional elements. The increase in computational efficiency for a single truss member is summarized in Table 8.4. Given that there is a distinct reduction in computational effort, all while maintaining an acceptable degree of accuracy, the capabilities of the NmT2 are to be further investigated in another case study which ramps up the complexity of the structure. For now, however, the three most important findings of this first case study are:

1. It is possible to embed data-driven elements in an active FEA analysis.
2. The NmT2 element shows a distinct decrease in computing time.
3. Despite some small discrepancies, the NmT2 element manages to maintain solution accuracy relative to FEA models meshed with traditional elements.

Table 8.4: Computational efficiency of traditional FEA elements versus a singular NmT2 element for case study 1.

	Average computing time [s]		
Loading case	T2D2 or B22	NmT2	**Gain in computational efficiency** [%]
Tension	39.68	24.48	62.13
Compression	85.47	22.24	284.22

9 Case Study 2: V-Shaped Structure

The second case study consists of what is considered a *bridging structure*. This refers to a kind of structure that builds on the first case study in a minimal form, but introduces vital components present when analyzing more complex multi-truss structures. Such components include joints, angular loading, and differential deformation. The inclusion of joints acknowledges that the structure has multiple members connected at their extremities. Angular loading occurs when the applied load is not coincident with the local coordinate system of a truss member and is therefore not aligned along a truss' axis. Finally, differential deformation arises when multiple members undergo opposite kinds of deformation. For example, certain loading schemes might induce tensile deformation in some truss members and compressive deformation in others. A V-shaped structure composed of two truss members represents a suitable model capable of demonstrating the contribution of all of these components.

An overview of the V-shaped structure is shown in Figure 9.1. As may be seen, the two truss members are joined at one extremity and simply-supported at the others. Displacement loading is applied at lower-most node inducing deformation in both truss members and therefore two reaction forces: $f^{(1)}_{e,\text{int}_1}$ and $f^{(2)}_{e,\text{int}_1}$. This chapter considers three kinds of loading:

1. Downward loading ($\Delta U_x = 0, \Delta U_y < 0$): both truss members will be in tension.
2. Upward loading ($\Delta U_x = 0, \Delta U_y > 0$): both truss members will be in compression.
3. Horizontal loading ($\Delta U_x > 0, \Delta U_y = 0$): one truss member will be in tension and the other in compression.

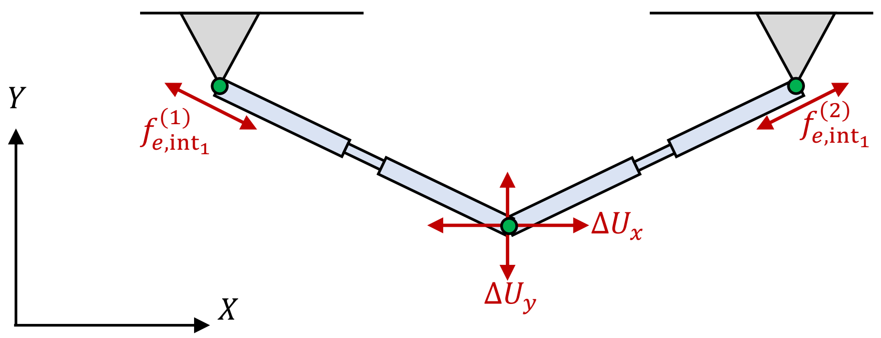


Figure 9.1: Case study 2: V-shaped structure of two truss members and loaded at its lower-most node.

The neuromorphic elements should be robust enough to model all three scenarios with sufficient accuracy relative to a model built with traditional FEA elements. This case study represents a crucial step in the development process of the NmT2 elements. If deemed successful, then more complex structures may be considered.

Regarding the analysis preparation, the same method as defined in the previous case study will be used for the V-shaped structure. A specifically developed program will generate two input files for the same structure, one using traditional FEA elements and the other using neuromorphic elements. The model using neuromorphic elements will consist of two NmT2 elements, each modelling a single truss member. The model using traditional FEA elements will possess far more depending on the loading scheme.

9.1. Results

As before, the geometrical parameters of the truss members are randomly sampled within the boundaries of the parameter design space. Given the nature of the V-shaped structure, a few additional constraints have to be addressed:

1. Theoretically the properties of each truss member are potentially subject to change. To ensure simplicity and provide a truly incremental approach to assessing the development of the neuromorphic element, it is assumed that the geometrical properties of each truss member are identical.
2. The angle between the two truss members is also a variable. If constrained to a constant degree, the choice of the angle would have to investigated as well as its effect (if any) on the results of the analysis. It could also be considered inhibitive to the development of the neuromorphic element since its performance would be assessed under only one angular configuration. Therefore, the angle between the truss members will be kept random to expose the NmT2 elements to alternate angular joints. To permit this, the distance between the supports will remain constant at a value of 17.5mm, regardless of the parameters of the truss members.
3. The joint between the truss members at the lower-most node is a pinned connection. This is important to underline when meshing a structure with beam elements. The rotational degrees of freedom have to be disabled to compare the results with the NmT2 elements which do not possess rotational degrees of freedom.

Two of the above constraints could be argued to be conflicting. The first constraint (constant distance between supports) promotes the incremental development of the neuromorphic element by forcing both truss members to have the same geometrical configurations. The second constraint subjects the structure to a randomly defined angle between the truss members, thereby preventing any concrete analysis of the effect of the angular separation in the joint (if any). The varying angle is not expected to have any effect on the performance of the NmT2 elements since they are only concerned about the axial displacement. The first constraint, however, implies vertical symmetry which facilitates the analysis of the three loading scenarios.

9.1.1. Scenario 1: Downward loading

The first scenario consists of downward loading such that structure is vertically loaded in the negative y direction ($\Delta U_x = 0, \Delta U_y < 0$), which puts both truss members equally in tension. Figures 9.2a and 9.2b illustrate the undeformed and deformed structure respectively. Before proceeding, it is important to reiterate the boundaries that were imposed during the training process of the neuromorphic engine. The boundary of the axial displacement of the truss member was confined at $\pm 50\%$ of the length of the defective portion of the truss. In other words, $\Delta a_{e,\text{axial}} \in \pm 0.025 L_e$. Since the V-structure is loaded at an angle respective to the axis of each truss member, the maximum displacement for scenario 1 loading is:

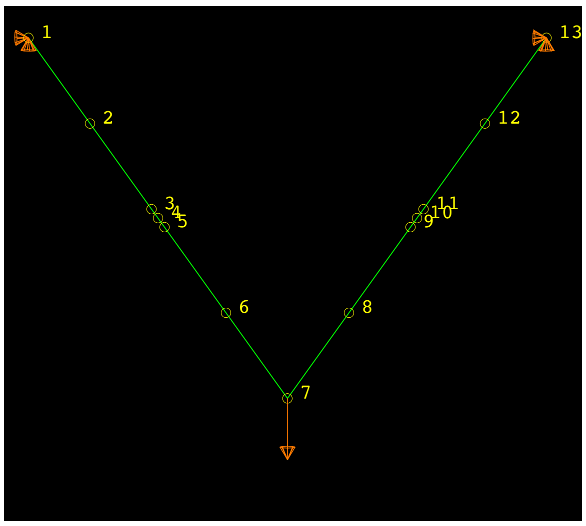
$$u_{y,\max} = -0.025 L_e \sin\left(\frac{1}{L_e}\sqrt{L_e^2 - \frac{b^2}{4}}\right) \quad \text{where: } (b = 17.5\text{mm}) \tag{9.1}$$

For the first loading scenario, meshing the traditional FEA structure with T2D2 is sufficient since each member will undergo axial tensile deformation. Just as in the first case study, a total of 30 structural configurations were randomly generated. Each structure was then meshed with standard FEA elements or simply 2 NmT2 elements. The only aspect of the process that changes relative to the first case study is the need for an additional post-processing step to compare the results between the FEA simulations. Since the performance of the neuromorphic element is assessed axially, then outputted results from the FEA simulation have to be transposed along its axis. Therefore, the following post-processing steps have to be incorporated:

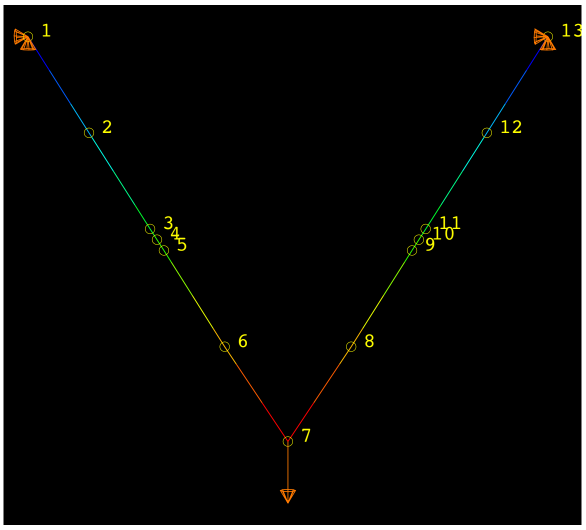
1. Given the angular nature of the V-shaped structure, the imposed downward displacement has to be converted into axial stretching of one of the two truss members. In other words, the data are extracted at the bottom-most node (where the load is applied), only ΔU_x and ΔU_y are obtained, which has to be transformed back into $\Delta a_{e,\text{axial}}$. This can be achieved by adapting Equation 9.1.

2. In addition to $\Delta a_{e,\text{axial}}$, the nodal reaction force at the boundary, $f^{(1)}_{e,\text{int}_1}$ or $f^{(2)}_{e,\text{int}_1}$, must be known. In this case, the axial reaction force corresponds to the magnitude of the reaction forces extracted (corresponding to RF1 and RF2 in ABAQUS).

For all 30 models, the tensile deformation curves of the nodal reaction force versus the axial displacement are reported and compared to one another. The lowest, highest, and average percentage



(a) Undeformed



(b) Deformed

Figure 9.2: V-shaped structure consisting of 2 truss members joined via a pin-connection with downward vertical loading to induce tensile deformation in both members, meshed with T2D2 elements. Heat map represents displacement magnitude.

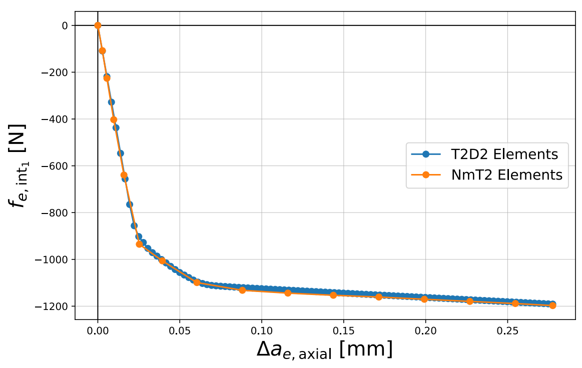
errors (denoted as ϵ_{min}, ϵ_{max} and $\bar{\epsilon}$ respectively) are also computed for each specimen relative to the traditional FEA models. Just as before, since these results are extracted from independent FEA analyses, it cannot be assumed that the data quantity will match. Therefore, the errors are extrapolated between the two closest data points. All the generated geometrical configurations, error analysis, and differences in computational time are illustrated in Table 9.1.

The first point of focus in the error analysis is always the R^2 values which are yet again considerably high: $R^2 > 0.99$ across all models. Although there is a slight decrease as the length of the specimen increases, it is negligible. Figure 9.3 on the following page shows a selection of 8 models from the 30 where the high degree of correlation is also visible. The absolute errors have slightly increased from the first case study, though remain relatively low, averaging around 2.4%. Regarding the computational time, the values have slightly increased relative to case 1 due to the presence of a second structural element, though the neuromorphic element's modelling capabilities still remain far more efficient than the traditional T2D2 elements.

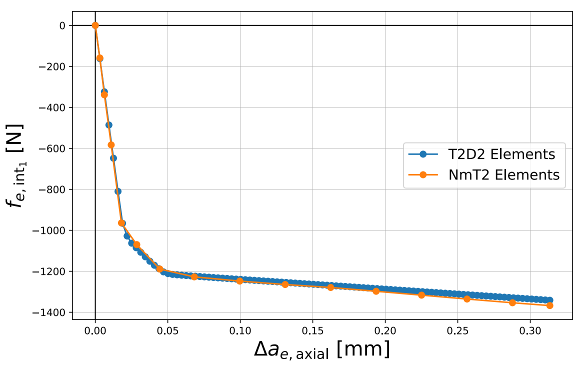
For the first loading scenario, the degree of correlation between the T2D2 elements and NmT2 is more than satisfactory. Additionally, the NmT2 elements reduce the computational time. They have no problem modelling the nonlinear regions due to plasticity as may be seen in Figure 9.3, and the oscillatory behaviour which was present in the first case study near the end of the deformation regime has vanished. This provides a confident foundation to proceed towards the second loading scenario.

Table 9.1: Error and temporal analysis for loading scenario 1 between traditionally meshed models using T2D2 elements and models using two NmT2 elements.

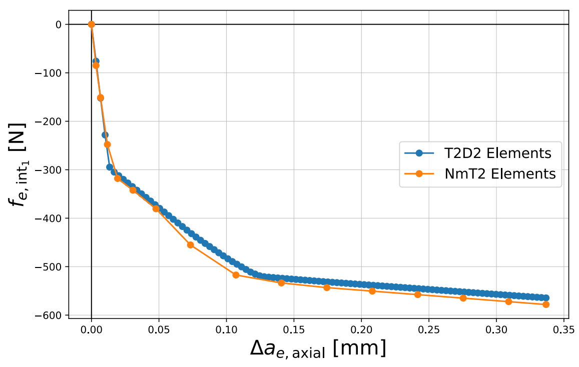
L_e [mm]	r_e [mm]	$r_{e,D}$ [mm]	$\epsilon_{\max}$ [%]	$\epsilon_{\min}$ [%]	$\bar{\epsilon}$ [%]	R^2 [-]	t_{T2D2} [s]	t_{NmT2} [s]
11.10	1.33	1.20	7.891	0.185	1.715	0.999	48.89	27.08
11.27	1.82	1.66	10.842	0.402	2.754	0.998	50.01	26.17
12.54	1.39	1.27	10.029	0.145	1.901	0.998	51.89	27.01
12.58	0.97	0.84	8.977	0.631	1.238	0.998	48.68	27.97
13.47	0.91	0.69	11.717	0.691	3.519	0.998	47.77	25.79
13.68	1.53	1.47	17.186	0.212	2.203	0.997	48.91	26.17
14.21	1.69	1.36	18.134	0.079	2.958	0.997	42.29	24.99
14.33	1.55	1.43	18.842	0.080	2.352	0.997	48.27	28.45
15.52	0.87	0.66	11.079	0.672	3.020	0.997	52.04	26.19
15.71	0.90	0.82	11.029	0.289	2.193	0.997	49.96	28.44
15.72	1.51	1.50	15.934	0.089	1.939	0.997	48.91	25.01
15.97	1.71	1.44	17.358	0.018	2.321	0.997	49.11	26.35
18.39	1.56	0.46	12.693	0.057	2.021	0.997	48.35	27.48
19.48	1.04	0.94	6.806	0.223	1.782	0.997	47.28	28.81
20.24	1.16	1.00	16.120	0.214	2.515	0.996	50.07	25.66
21.61	1.94	1.56	4.781	0.474	2.002	0.996	49.38	29.09
21.87	1.68	1.68	13.939	0.171	2.086	0.996	48.93	27.71
22.94	1.28	1.20	18.773	0.032	2.972	0.996	47.19	25.49
23.16	1.89	1.87	15.134	0.037	2.091	0.996	46.66	25.55
24.25	1.11	0.85	5.535	1.397	2.798	0.996	49.13	25.19
25.04	1.89	1.52	5.304	0.753	2.231	0.996	50.35	23.07
25.32	1.50	1.26	3.236	0.849	1.888	0.996	50.46	26.96
26.27	2.00	1.43	9.773	0.534	3.203	0.996	51.26	26.56
26.31	1.05	0.92	11.280	0.286	2.807	0.996	49.15	24.23
26.55	1.04	0.99	14.258	0.017	2.956	0.996	47.28	26.70
26.56	1.03	0.89	8.097	0.270	2.101	0.996	49.58	23.68
27.54	1.53	1.36	11.527	0.163	1.927	0.996	47.61	25.05
28.95	1.93	1.57	4.621	0.042	2.182	0.996	48.09	28.86
29.36	0.72	0.62	9.245	0.022	2.467	0.996	47.47	26.01
29.75	1.07	1.05	14.654	0.166	2.981	0.996	49.96	24.42
Average			11.493	0.307	2.371	0.997	48.83	26.34



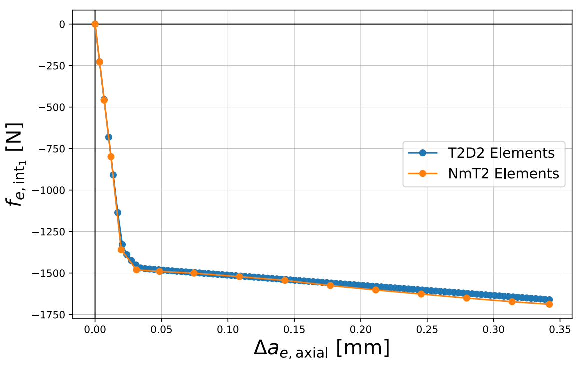
(a) $\{L_e, r_e, r_{e,D}\} = [11.10\text{mm}, 1.33\text{mm}, 1.20\text{mm}]$



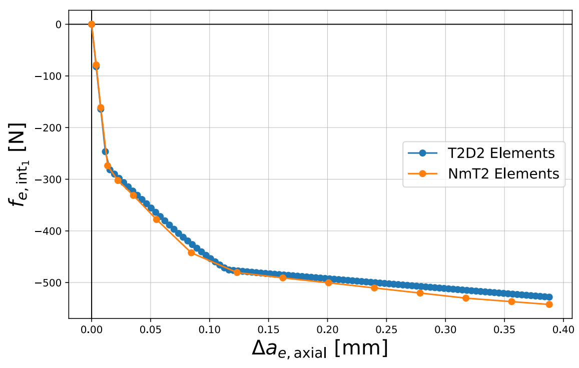
(b) $\{L_e, r_e, r_{e,D}\} = [12.54\text{mm}, 1.39\text{mm}, 1.27\text{mm}]$



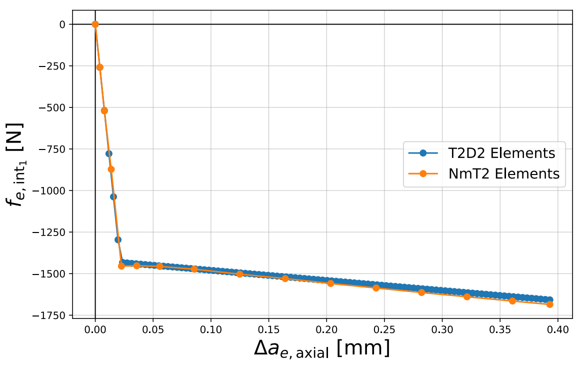
(c) $\{L_e, r_e, r_{e,D}\} = [13.47\text{mm}, 0.91\text{mm}, 0.69\text{mm}]$



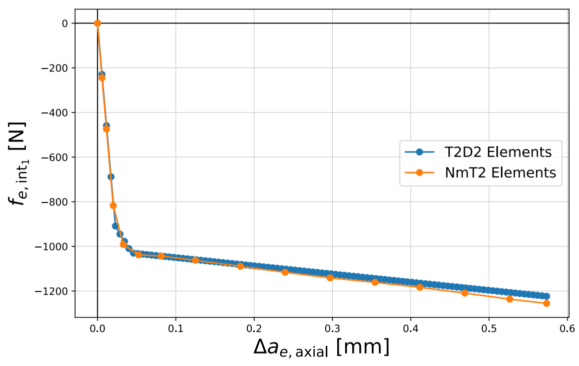
(d) $\{L_e, r_e, r_{e,D}\} = [13.68\text{mm}, 1.53\text{mm}, 1.47\text{mm}]$



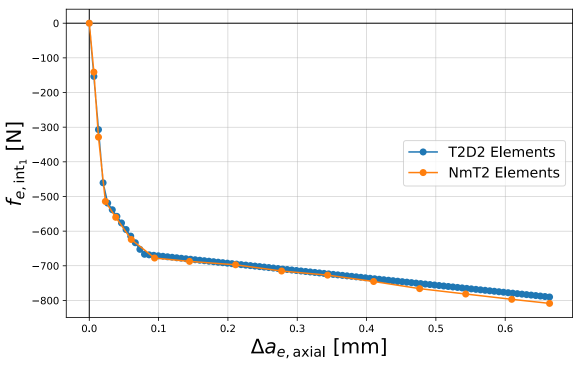
(e) $\{L_e, r_e, r_{e,D}\} = [15.52\text{mm}, 0.87\text{mm}, 0.66\text{mm}]$



(f) $\{L_e, r_e, r_{e,D}\} = [15.72\text{mm}, 1.51\text{mm}, 1.50\text{mm}]$



(g) $\{L_e, r_e, r_{e,D}\} = [22.94\text{mm}, 1.28\text{mm}, 1.20\text{mm}]$



(h) $\{L_e, r_e, r_{e,D}\} = [26.56\text{mm}, 1.03\text{mm}, 0.80\text{mm}]$

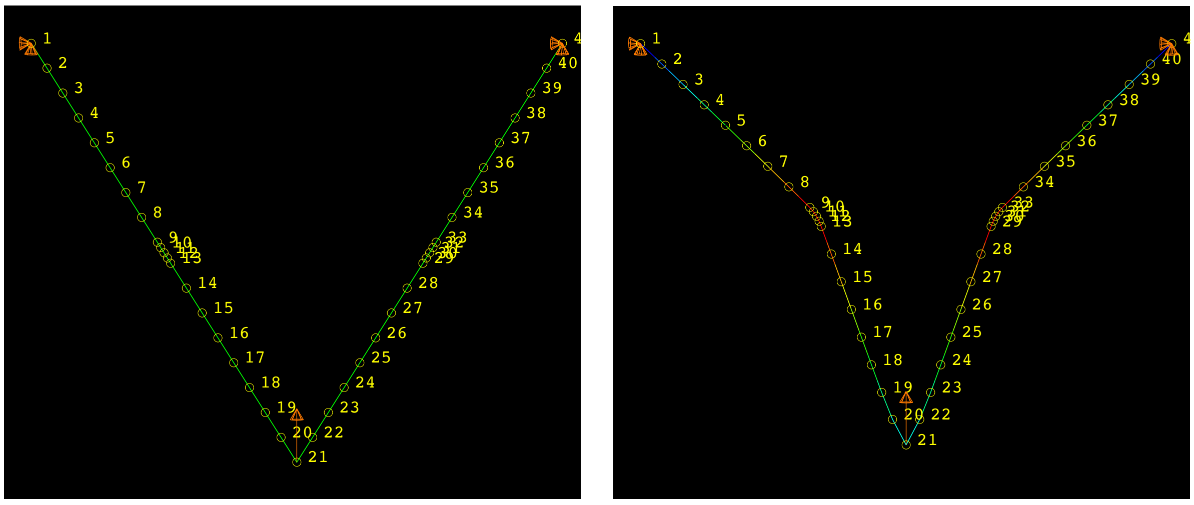
Figure 9.3: Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (T2D2), and the same models meshed with two NmT2 elements (shown in orange) for loading scenario 1 applied to a V-shaped structure (2 truss members).

9.1.2. Scenario 2: Upward loading

The second scenario consists of upward loading such that both truss members are in compression ($\Delta U_x = 0, \Delta U_y > 0$). This will lead to both members buckling as may be seen in Figures 9.4a and 9.4b. As in the downward loading scenario, the vertical displacement loading is translated to axial shortening ($\Delta a_{e,\text{axial}}$) by adapting Equation 9.1. It is worth reiterating that since compression to the point of post-buckling is investigated at this stage, the T2D2 elements are swapped for B22 elements which promote deformation out of the axial plane. Additionally, more FEA elements are required to ensure mesh and solution convergence; this will not be investigated here to not detract from the overall goal of this chapter. However, mesh convergence checks were conducted in the same fashion as described in the data-generation phase of the thesis in Chapter 5. Figures 9.4a and 9.4b are simply illustrative and should not be interpreted as a reference regarding the number of B22 elements which varies among models.

Once again, the absolute errors, R^2 metrics and computational times are summarized in Table 9.2, and the deformation profiles for 8 models selected from the 30 are shown in Figure 9.5. As expected, the overall errors are slightly larger than in the first scenario, though still remain quite small. The R^2 metric is again always greater than 0.99 indicating a high degree of correlation is maintained despite the highly nonlinear nature of the post-buckling regime. The NmT2 accurately capture the nonlinear degradation of the structure's integrity in the post-buckling regime. The close approximation of the NmT2 elements relative to the B22 elements is remarkable.

In the case of absolute errors, they are higher, as to be expected from the decrease in R^2 metric. The maximum error arises from two sources. The first source is the buckling peak where the neuromorphic elements tend to slightly overshoot as may be see in Figure 9.5c. The second source of discrepancies occurs in the post-buckling regime, at second minor peak visible in Figures 9.5b, 9.5c, 9.5e and 9.5g, where the NmT2 either slightly over or underestimates the peak. In any case, the absolute errors still remain quite low and average just over 3% which is more than acceptable. Relative to the first case study, the average error is 3 times higher.



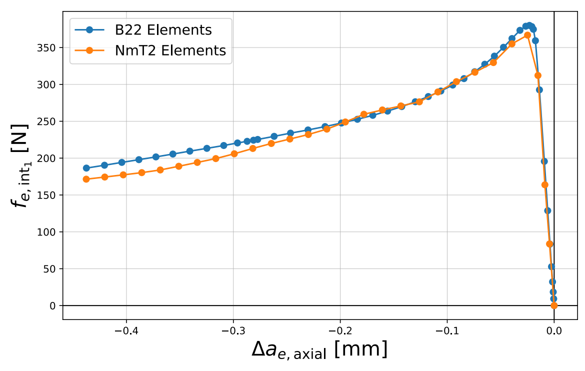
(a) Undeformed (b) Deformed

Figure 9.4: V-shaped structure consisting of 2 truss members joined via a pin-connection with upward loading to induce buckling in both members, meshed with B22 elements. Heat map represents displacement magnitude.

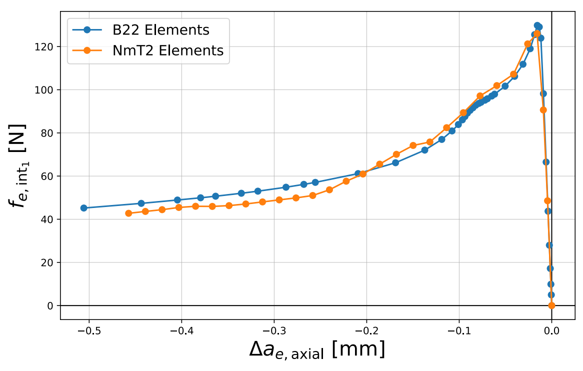
From a computational efficiency standpoint, the neuromorphic elements maintain a similar efficiency as before averaging 22 seconds across all 30 models. This is starting to be substantial when compared to the B22 elements which require almost 2 full minutes to converge to a solution. The benefits of data-driven elements are becoming apparent as the complexity of the structure increases, provided that the small deviations are acceptable. The next step is to test a loading scenario that triggers differential loading in the truss members.

Table 9.2: Error and temporal analysis for loading scenario 2 between traditionally meshed models using B22 elements and models using two NmT2 elements.

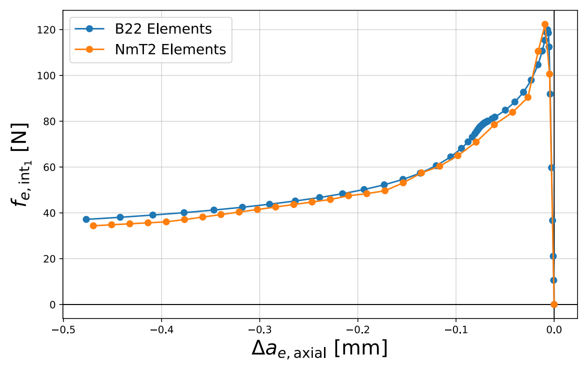
L_e [mm]	r_e [mm]	$r_{e,D}$ [mm]	$\epsilon_{\max}$ [%]	$\epsilon_{\min}$ [%]	$\bar{\epsilon}$ [%]	R^2 [-]	t_{B22} [s]	t_{NmT2} [s]
10.44	1.54	1.49	17.279	0.538	4.000	0.991	102.81	26.14
10.49	1.57	1.38	13.734	0.737	3.404	0.991	111.74	22.81
11.39	1.63	1.15	6.056	0.557	3.122	0.991	115.27	19.26
13.34	1.62	1.52	2.886	0.151	1.460	0.992	106.24	20.08
15.12	1.63	1.62	3.648	0.033	2.027	0.992	100.27	21.35
15.82	1.92	1.72	3.875	0.015	1.196	0.993	104.35	26.68
17.31	0.88	0.84	8.899	0.039	4.251	0.993	106.63	25.98
18.09	0.65	0.53	11.168	0.316	6.238	0.993	103.33	22.84
18.32	1.31	1.23	4.536	0.200	1.218	0.993	101.12	23.49
18.57	0.99	0.49	8.678	0.176	4.962	0.993	117.75	22.37
18.72	1.20	1.15	5.466	0.219	2.585	0.993	119.26	22.39
19.60	1.14	0.90	8.923	0.016	3.261	0.993	113.25	23.04
21.79	1.90	1.69	2.165	0.031	0.987	0.993	106.82	24.25
22.05	0.87	0.74	6.009	0.018	1.933	0.993	109.25	21.73
22.83	1.24	0.89	6.500	0.135	2.729	0.993	101.16	22.03
24.40	1.38	1.31	6.642	0.060	2.329	0.993	104.46	22.84
24.39	0.73	0.50	12.12	0.032	4.676	0.993	111.15	21.96
24.78	0.56	0.50	18.828	0.198	5.558	0.993	121.42	21.97
25.40	1.34	0.84	10.47	0.114	3.550	0.993	116.73	21.02
25.53	1.56	1.44	6.571	0.207	3.829	0.993	113.35	19.78
27.56	1.94	1.84	5.319	0.007	2.715	0.992	108.85	19.99
27.74	0.75	0.60	10.791	0.096	2.213	0.992	107.74	23.01
27.91	1.30	1.42	5.871	0.233	2.508	0.992	114.85	24.45
27.92	1.29	1.12	6.756	0.114	2.343	0.992	105.52	21.28
28.06	1.14	0.74	8.331	0.006	5.010	0.992	101.12	19.35
28.68	0.78	0.63	12.219	0.048	2.895	0.992	119.93	19.72
28.71	1.04	0.96	6.705	0.024	3.784	0.992	118.85	18.97
28.86	1.08	0.80	7.951	0.030	3.246	0.992	117.83	25.81
29.42	0.74	0.73	9.803	1.154	5.907	0.992	114.47	21.26
29.96	1.29	1.19	6.097	1.157	3.542	0.992	108.83	24.01
Average			8.143	0.222	3.249	0.992	110.15	22.33



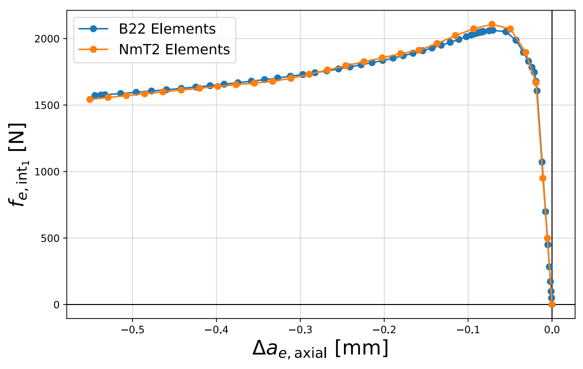
(a) $\{L_e, r_e, r_{e,D}\} = [17.31\text{mm}, 0.88\text{mm}, 0.84\text{mm}]$



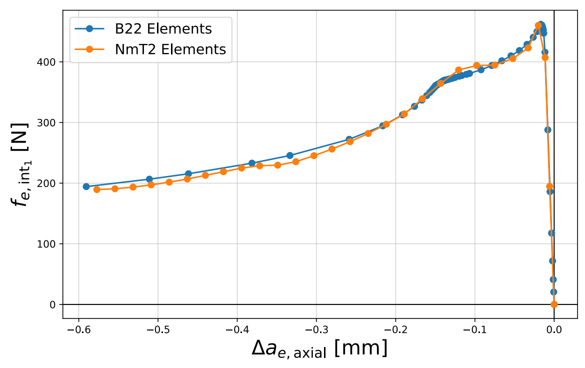
(b) $\{L_e, r_e, r_{e,D}\} = [18.09\text{mm}, 0.65\text{mm}, 0.53\text{mm}]$



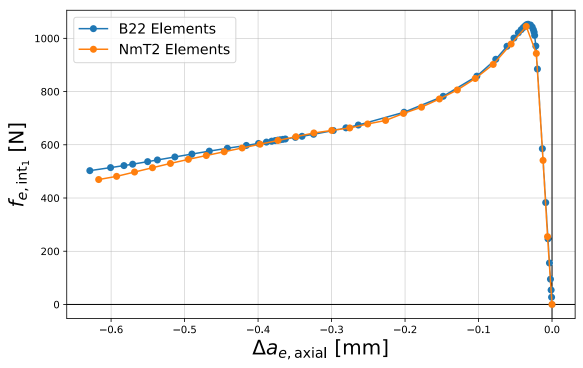
(c) $\{L_e, r_e, r_{e,D}\} = [18.57\text{mm}, 0.99\text{mm}, 0.49\text{mm}]$



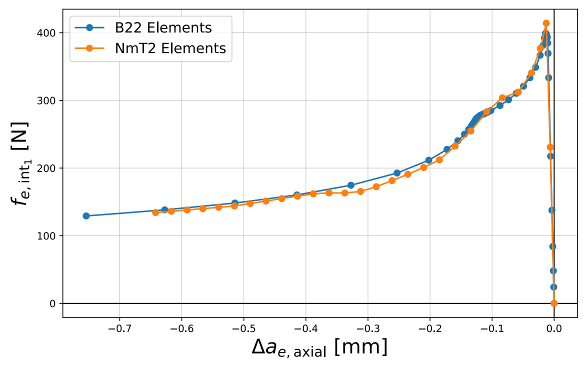
(d) $\{L_e, r_e, r_{e,D}\} = [21.79\text{mm}, 1.90\text{mm}, 1.69\text{mm}]$



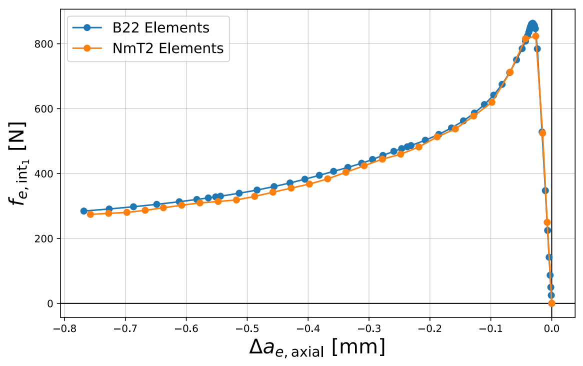
(e) $\{L_e, r_e, r_{e,D}\} = [22.83\text{mm}, 1.24\text{mm}, 0.89\text{mm}]$



(f) $\{L_e, r_e, r_{e,D}\} = [24.40\text{mm}, 1.38\text{mm}, 1.31\text{mm}]$



(g) $\{L_e, r_e, r_{e,D}\} = [25.40\text{mm}, 1.34\text{mm}, 0.84\text{mm}]$



(h) $\{L_e, r_e, r_{e,D}\} = [29.96\text{mm}, 1.29\text{mm}, 1.19\text{mm}]$

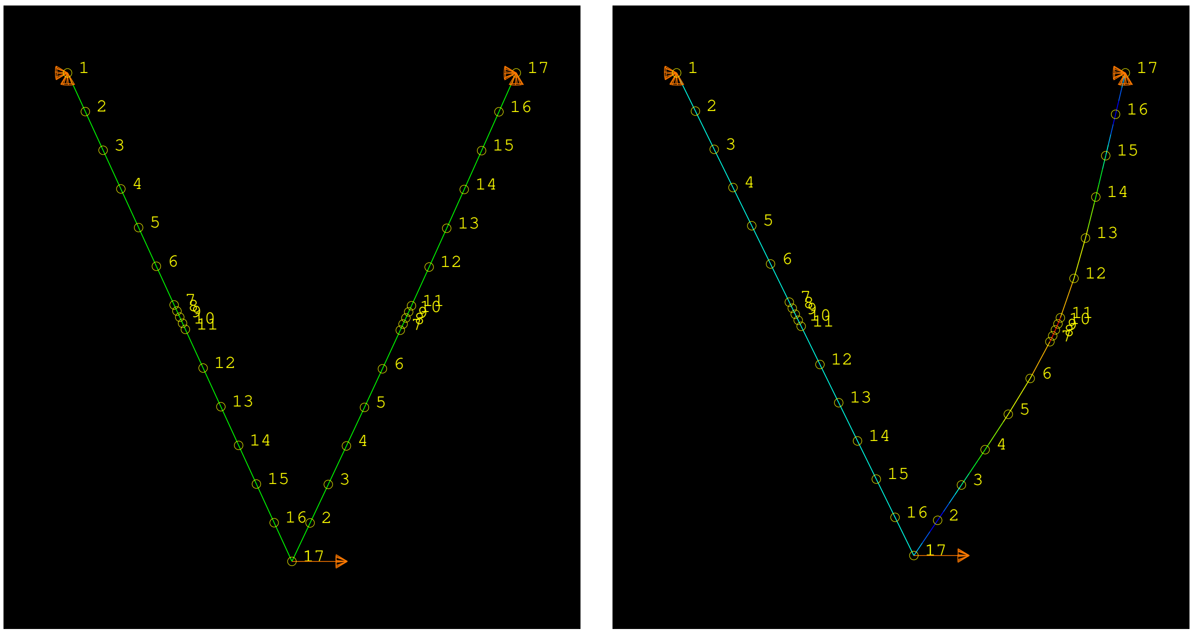
Figure 9.5: Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (B22), and the same models meshed with two NmT2 elements (shown in orange) for loading scenario 2 applied to a V-shaped structure (2 truss members).

9.1.3. Scenario 3: Horizontal loading

The final scenario consists of loading the structure horizontally, to the right ($\Delta U_x > 0, \Delta U_y = 0$). This will induce differential loading in the truss members: the right member will be under compressive loading whereas the other will be under tension as may be seen in Figures 9.6. The error analyses from the previous scenarios should not overshadow the fact that the traditional FEA processes are being matched by a data-driven element. This final loading scenario will evaluate how it performs when used in a structure where not all structural members are loaded in the same direction.

Just as in the two previous cases, the error analysis for all 30 randomly generated models is present in Table 9.3, and the deformation plots for 8 models selected from the 30 are displayed in Figure 9.7. However, given that one truss member will be in tension and the other in compression, the graphical illustrations are slightly different. As before, the magnitudes of the reaction forces at the two boundary nodes are extracted, however instead of using just one, the results form both truss models are plotted in Figure 9.7. Plots with a marker that has a black filling are extracted from the boundary node whose structural member is in tension. The reason behind this differentiation is to verify that both tensile and compressive deformation are occurring simultaneously in the structure.

The first noteworthy aspect of the deformation is that, unlike previous cases, the compression and tension side are perfectly mirrored. Given that buckling cripples the structural integrity, the reaction forces at both nodes are expected to be equal and opposite to balance the statically determinant system. When looking at the error analysis in Table 9.3, the variation in R^2 metric is minimal and always greater than 0.98 indicating a very good degree of correlation between the B22 and NmT2 elements. This is slightly less than in the upward loading scenario where all the R^2 values were greater than 0.99. The small drop in modelling capabilities can be attributed to the differential deformation induced by the horizontal loading. As always, the absolute errors are less indicative



(a) Undeformed (b) Deformed

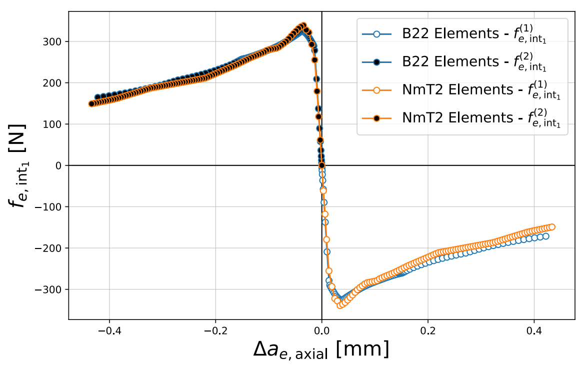
Figure 9.6: V-shaped structure consisting of 2 truss members joined via a pin-connection with horizontal loading to induce differential loading in both members, meshed with B22 elements. Heat map represents displacement magnitude.

than the R^2 metric, though it is noteworthy that the average error, $\bar{\epsilon}$ remains less than 10%. Overall, the discrepancies remain very small. The NmT2 elements manage to accurately model the structural behaviour including the nonlinear region, which can be qualitatively seen in Figure 9.7.

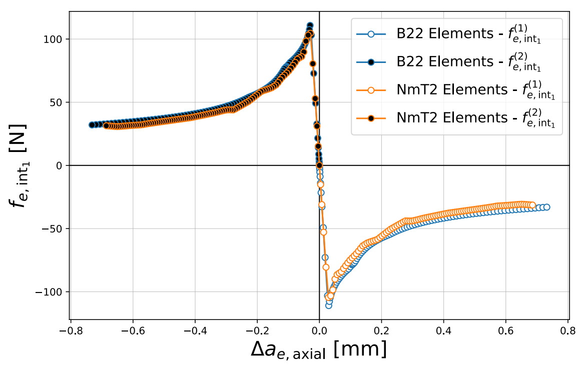
In addition to the nodal force response curves, there is another interesting phenomenon which occurs in this horizontal loading scenario that was not present in the first two: the global displacement of the structure. Due to the structural deformation, the bottom-most node will move to the right in the direction of the applied load, but also upwards: visible in Figure 9.6b. The vertical transition is not imposed, but induced from the structural deformation of both truss members and is part of the final displacement field of the FEA analysis. Up until now, the displacement fields of the previous scenarios have not been investigated since no alternate displacement was induced due to structural deformation.

Table 9.3: Error and temporal analysis for loading scenario 3 between traditionally meshed models using B22 elements and models using two NmT2 elements.

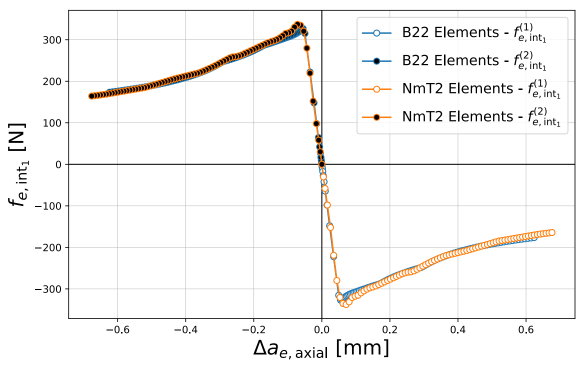
L_e [mm]	r_e [mm]	$r_{e,D}$ [mm]	$\epsilon_{\max}$ [%]	$\epsilon_{\min}$ [%]	$\bar{\epsilon}$ [%]	R^2 [-]	t_{B22} [s]	t_{NmT2} [s]
11.83	0.85	0.73	14.219	0.059	3.445	0.988	142.36	23.06
14.83	1.36	1.03	8.662	0.300	3.088	0.982	139.56	22.61
15.12	1.40	1.23	8.945	0.006	2.673	0.986	129.95	24.07
15.42	0.55	0.48	10.528	0.189	6.217	0.986	150.51	25.55
16.23	0.75	0.46	11.521	0.006	4.423	0.986	149.96	20.01
17.13	0.64	0.53	19.959	0.339	7.461	0.986	135.47	19.96
17.54	1.00	0.86	8.018	0.031	1.590	0.986	137.96	27.83
18.51	0.91	0.77	11.234	0.014	1.862	0.986	151.45	20.94
19.35	1.10	0.71	10.487	0.004	2.385	0.987	148.76	22.23
19.43	0.92	0.61	19.034	0.055	6.012	0.986	144.42	24.41
20.59	0.75	0.63	16.842	0.357	8.685	0.985	149.92	26.68
21.28	1.13	0.69	10.807	0.012	2.387	0.985	140.06	24.41
21.54	1.59	1.51	17.602	0.022	1.622	0.989	129.93	20.09
21.98	1.15	0.89	16.476	0.060	2.609	0.989	132.59	27.75
22.56	1.11	1.04	19.682	0.055	2.048	0.989	133.39	28.86
22.79	1.50	1.43	20.215	0.089	1.905	0.989	139.72	26.64
22.98	1.30	1.28	20.101	0.025	1.994	0.989	152.15	22.28
24.91	1.39	1.27	21.049	0.015	2.149	0.989	140.05	23.39
25.15	0.77	0.48	16.770	0.068	5.257	0.989	144.46	20.04
25.22	0.75	0.58	16.031	0.084	4.138	0.989	128.86	26.64
25.86	0.71	0.48	26.762	0.202	4.803	0.989	129.46	19.75
26.19	1.05	1.01	36.151	0.055	3.382	0.989	130.08	26.62
27.44	0.75	0.50	23.015	0.001	3.177	0.989	144.62	23.31
27.52	1.21	0.99	20.353	0.142	3.745	0.989	143.34	26.68
27.61	0.67	0.53	26.852	0.118	4.658	0.989	139.96	24.87
27.94	0.57	0.51	30.532	0.082	6.562	0.989	133.32	23.99
28.12	1.70	1.65	21.452	0.008	2.377	0.989	150.01	24.24
28.13	1.92	1.89	25.046	0.487	3.843	0.990	147.96	23.95
28.62	0.89	0.84	27.680	0.388	4.381	0.989	148.28	27.70
29.28	1.29	1.24	24.091	0.004	2.680	0.989	142.25	24.46
Average			18.671	0.109	3.719	0.988	141.03	24.10



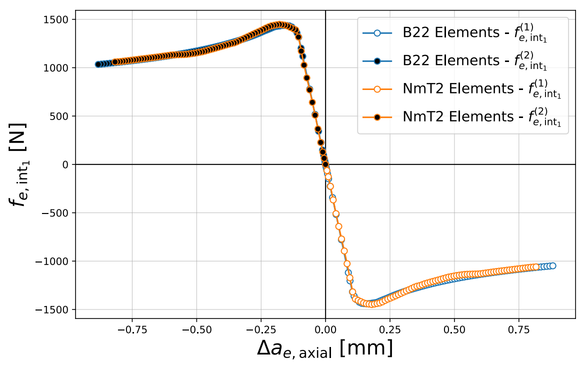
(a) $\{L_e, r_e, r_{e,D}\} = [11.83\text{mm}, 0.85\text{mm}, 0.73\text{mm}]$



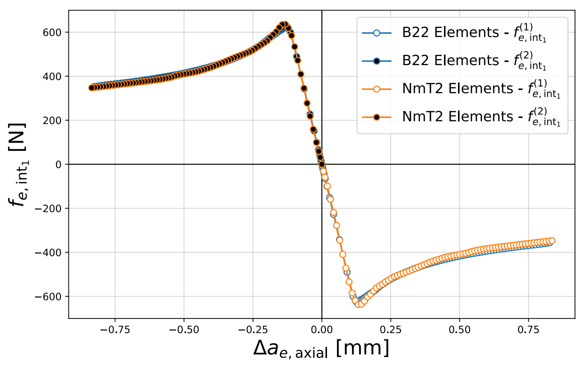
(b) $\{L_e, r_e, r_{e,D}\} = [15.42\text{mm}, 0.55\text{mm}, 0.48\text{mm}]$



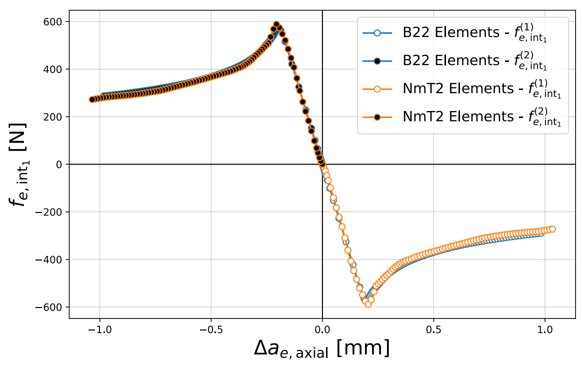
(c) $\{L_e, r_e, r_{e,D}\} = [18.51\text{mm}, 0.91\text{mm}, 0.77\text{mm}]$



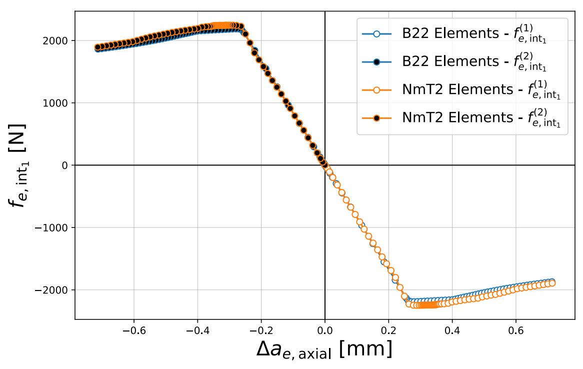
(d) $\{L_e, r_e, r_{e,D}\} = [21.54\text{mm}, 1.59\text{mm}, 1.51\text{mm}]$



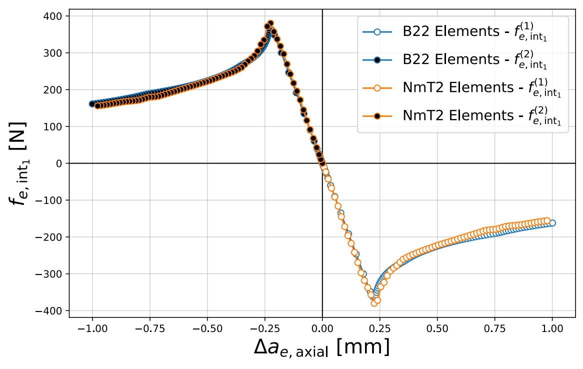
(e) $\{L_e, r_e, r_{e,D}\} = [22.56\text{mm}, 1.11\text{mm}, 1.04\text{mm}]$



(f) $\{L_e, r_e, r_{e,D}\} = [26.19\text{mm}, 1.05\text{mm}, 1.01\text{mm}]$

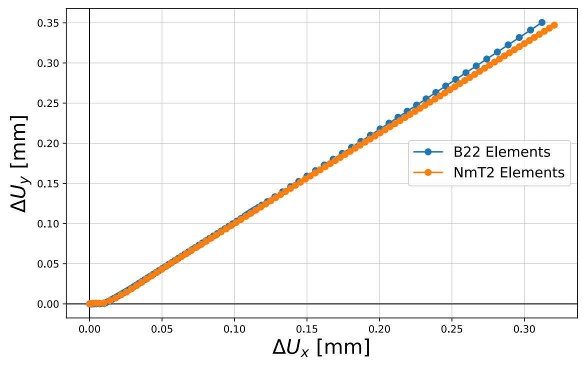


(g) $\{L_e, r_e, r_{e,D}\} = [28.13\text{mm}, 1.92\text{mm}, 1.89\text{mm}]$

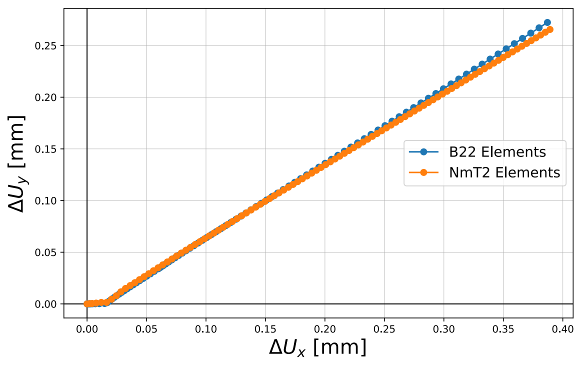


(h) $\{L_e, r_e, r_{e,D}\} = [28.62\text{mm}, 0.89\text{mm}, 0.84\text{mm}]$

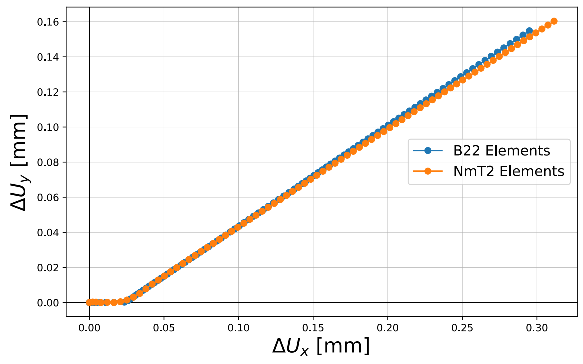
Figure 9.7: Comparison between a selection of randomly generated FEA models within the prescribed parameter design space using traditional FEA elements (B22), and the same models meshed with two NmT2 elements (shown in orange) for loading scenario 3 applied to a V-shaped structure (2 truss members).



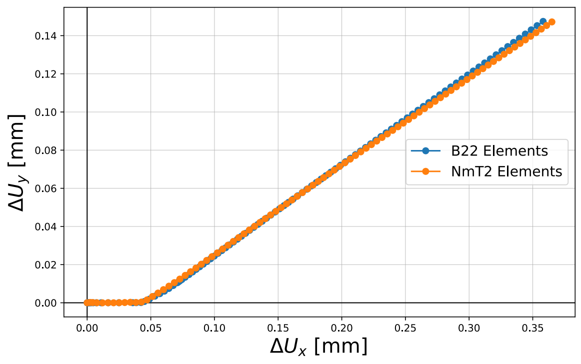
(a) $\{L_e, r_e, r_{e,D}\} = [11.83\text{mm}, 0.85\text{mm}, 0.73\text{mm}]$



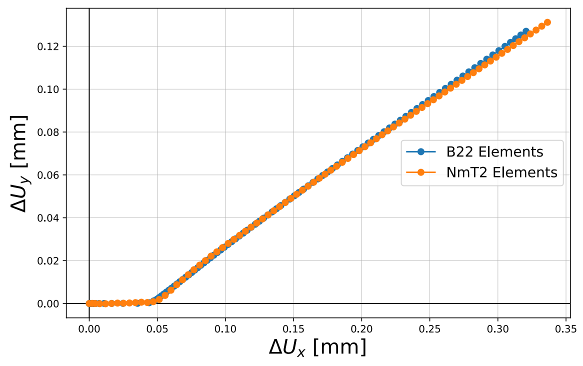
(b) $\{L_e, r_e, r_{e,D}\} = [15.42\text{mm}, 0.55\text{mm}, 0.48\text{mm}]$



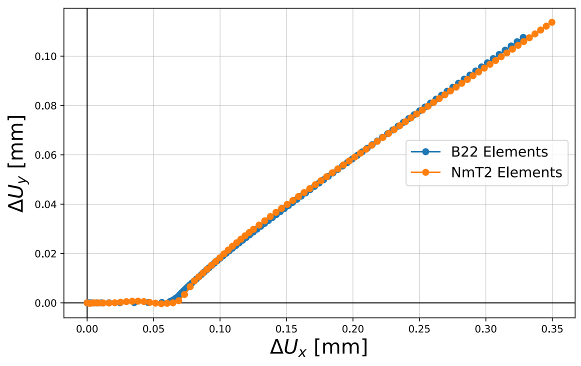
(c) $\{L_e, r_e, r_{e,D}\} = [18.51\text{mm}, 0.91\text{mm}, 0.77\text{mm}]$



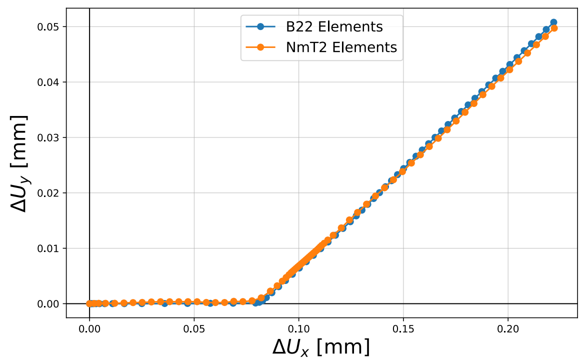
(d) $\{L_e, r_e, r_{e,D}\} = [21.54\text{mm}, 1.59\text{mm}, 1.51\text{mm}]$



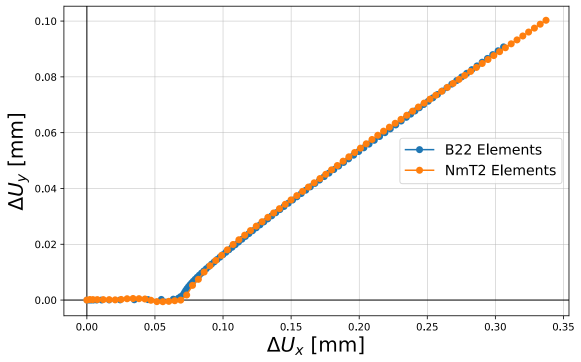
(e) $\{L_e, r_e, r_{e,D}\} = [22.56\text{mm}, 1.11\text{mm}, 1.04\text{mm}]$



(f) $\{L_e, r_e, r_{e,D}\} = [26.19\text{mm}, 1.05\text{mm}, 1.01\text{mm}]$



(g) $\{L_e, r_e, r_{e,D}\} = [28.13\text{mm}, 1.92\text{mm}, 1.89\text{mm}]$



(h) $\{L_e, r_e, r_{e,D}\} = [28.62\text{mm}, 0.89\text{mm}, 0.84\text{mm}]$

Figure 9.8: Displacement profiles between B22 and NmT2 elements of the structures extracted from Figure 9.7 of the bottom-most node in the V-structure showing the correlation between the horizontal and vertical displacement (U_x and U_y) due to the applied horizontal load.

Figure 9.8 shows the displacement profile of the lower node on the V-shaped structure for all the models plotted in Figure 9.7. These displacement profiles illustrate the induced vertical displacement (ΔU_y) of the loaded node due to the structural deformation that occurs from the imposed horizontal displacement (ΔU_x). Just as with the nodal force response curves, the correlation between the NmT2 elements and the FEA elements is excellent for all cases. The plateau at the beginning of the plots represents the linear elastic deformation regime of the structure during which the right truss member has not buckled yet. When the right truss member finally buckles, the bottom node starts to move up; i.e: $\Delta U_y > 0$. The NmT2 elements manage to capture the full displacement of the bottom node in addition to the transition region which is of upmost importance.

9.2. Closure

The purpose of case study 2 was to test the capabilities of the neuromorphic element in an incremental manner and introduce multi-element interaction in an active FEA analysis. It is therefore important to reiterate some of the significant observations of this second case study:

1. It is indeed possible to build multiple data-driven elements in a live FEA setting.
2. The computational efficiency of neuromorphic elements is consistently higher than that of traditional FEA elements, especially in the case of nonlinear phenomena such as multi-member buckling or differential loading.
3. Neuromorphic elements also circumvent the need for an eigenvalue analysis on the structure, which traditional FEA methods require to model post-buckling.

Overall, case study 2 can be considered positive as it demonstrated that the neuromorphic elements remained robust against ramping-up the structural complexity, and maintained a far greater computational efficiency than traditional methods. The key gains in computational efficiency are summarized in Table 9.4. As shown throughout all three loading scenarios, the gains in computational efficiency cannot be neglected, and is the biggest strength of the neuromorphic elements.

It should also be noted that computational efficiency is measured in terms of the computing time required for an FEA model to converge. However, with this second case study, there is an additional source of computational gain which is provided through the neuromorphic element, which is the time required to build the FEA model. This was not a problem in the first case study since a single horizontal truss member is relatively simple to build in ABAQUS. However, throughout case study 2, it took far longer to build the FEA model with traditional elements, whether it be through an input file or the GUI. Naturally, this metric is not as concrete as the computing time metric since it depends on the user's experience. But it shows that using NmT2 elements benefits from an accelerated computing time while also avoiding the build time of the FEA model.

Table 9.4: Computational efficiency measured in terms of computing/processing time of traditional FEA elements versus two NmT2 elements for case study 2.

	Average computing time [s]		
Loading case	T2D2 or B22	NmT2	**Gain in computational efficiency** [%]
Scenario 1	48.83	26.34	85.38
Scenario 2	110.15	22.33	393.28
Scenario 3	141.03	24.10	485.19

10

Case Study 3: Howrah Structure

The third and final case study is required to ramp-up the complexity of the overall structure to test more advanced capabilities of the neuromorphic element. The previous two case studies have shown that the NmT2 element is capable of modelling tensile and compressive deformation with highly nonlinear phenomena in a live FEA setting, and is not hindered by multi-element interaction. There is an evident gain in computational efficiency, especially for nonlinear deformation which occurs when a truss member is loaded under compression. The main requirements of the third case study are such that they should increase the overall complexity to the point where the structure is no longer verifiable by hand, meaning that it can be considered a validation study. The arrangement of truss members should also be such that differential loading always occurs, in every truss member. With these criteria in mind, the Howrah bridge was chosen as a source of inspiration.

Bridges are a wonderful example of multi-truss structures which facilitate day-to-day lives. Perhaps one of the most impressive is the Howrah bridge located in Kolkata, India. Commissioned in 1943, the Howrah bridge is made of pure steel and spans 700m over the Hooghly river in West Bengal shown in Figure 10.1. The sixth largest multi-truss bridge in the world supports over 100,000 vehicles and 150,000 pedestrians each day, which is 4 times the volume it carried when it opened in 1946. The structural loads are so significant that the bridge has started to sag over the years to the point where it is now illegal to slow down slightly or to stop on the bridge to take a picture. The multi-truss pattern used throughout is not fully repetitive, but changes as it nears the extremities of the bridge. There is however a distinct repetitive unit in the middle portion of the bridge, which is a

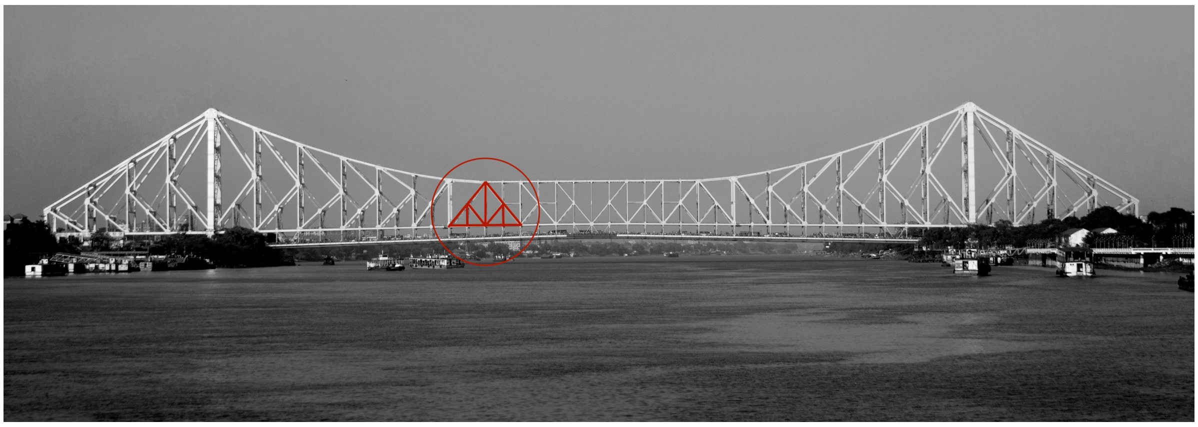


Figure 10.1: The Howrah bridge in Kolkata, spanning over the Hooghly river [97].

triangular unit composed of 13 truss members shown highlighted in red in Figure 10.1. This is the chosen structural configuration for the third case study.

In Chapter 4, the chosen context for the neuromorphic element was biomimicry for small steel structures that could be 3D printed as a singular unit. Using a component from a bridge introduces more complexity, an order of magnitude higher. However, this does not detract from the initial goal as such a structure could just as easily arise in a small structure organic lattice. The Howrah bridge is simply an example found in real-life which helps us connect the abstract development of the NmT2 element to a real-world structure. The final application for the neuromorphic element has not changed.

An idealized diagram of the structure for the purposes of this case study is shown in Figure 10.2. It is pinned at node 1 and simply supported at node 5. A downward displacement load is applied at node 3, which is representative of the loads facing the entire bridge. The remaining nodes and elements are numbered 1 through 8 and 1 through 13 respectively, and the lengths and radii are bounded by symmetry ($L_1 = L_3, L_2 = L_4, L_5 = L_8, ...$). Furthermore, the structure is such that the outer members possess the same radii; i.e.: $r_1 = r_2 = r_3 = r_4$ and $r_5 = r_6 = r_7 = r_8$. Just as before, structural defects in terms of defect radii $R_{e,D}$ can be applied to each member. The aforementioned geometric boundaries are such that the structure can retain some simplicity and avoid large radial differences between adjacent structural members in line with one another. From here, two scenarios will be investigated:

1. No defect seeding. To ensure the model behaves correctly with the NmT2 elements, it is first tested with no radial defects (meaning that $r_{e,D} = r_e$).

2. Random defect seeding. In reality, structural defects do not occur symmetrically throughout the structure. They are randomly distributed throughout the structural components. This will be the most complex configuration to which the NmT2 element will be subjected, and this configuration concludes the case study series of this thesis.

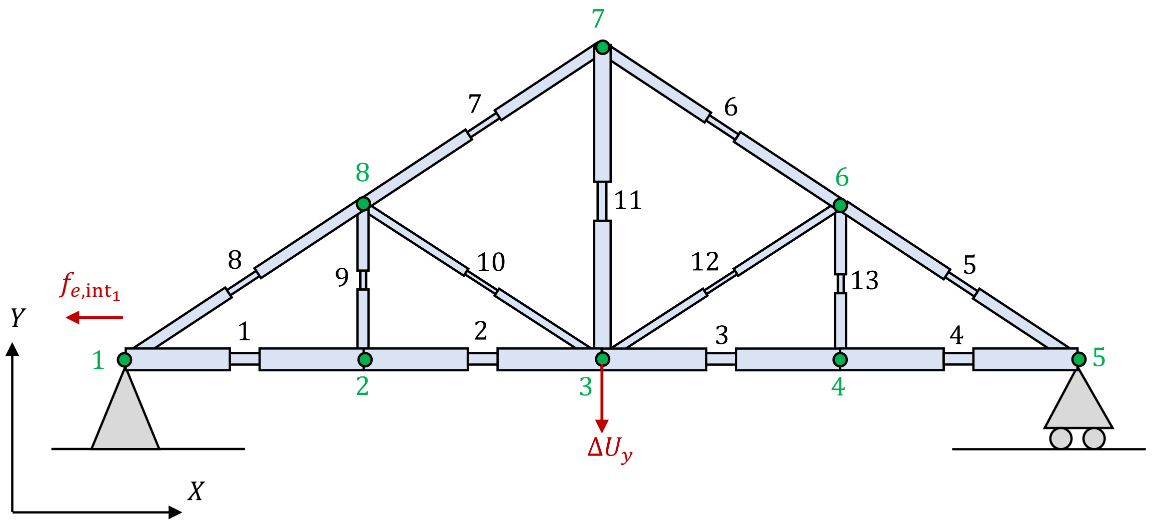


Figure 10.2: Overview of the Howrah structure consisting of 13 truss members. The structure is pinned at node 1 and simply supported at node 5. A downwards displacement load (ΔU_y) is applied at node 3. Throughout this case study, load-displacement curves are built by plotting the reaction force at node 1, f_{e,int_1}, versus ΔU_y.

10.1. Results

Given the complexity of the Howrah structure, a detailed analysis will not be conducted on 30 randomly generated models, but rather only 3. Regarding the random selection of the geometry, given the symmetry boundary, only L_1, L_2 and L_3 are required which will by extension define the lengths of all the other truss members. A check is performed on all the lengths of a randomly generated model such that they all lie within the prescribed parameter design space on which the NmT2 element was trained. Concerning the radii, they are randomly sampled within their respective domain taking into account the applied radial constraints. Once the model is constructed, it is loaded downward at node 3 through U_y. As explained previously, it is first important to make sure the FEA model and NmT2 element behave properly, which is why the defect is left out for the first scenario, thereby reducing the complexity of the structure.

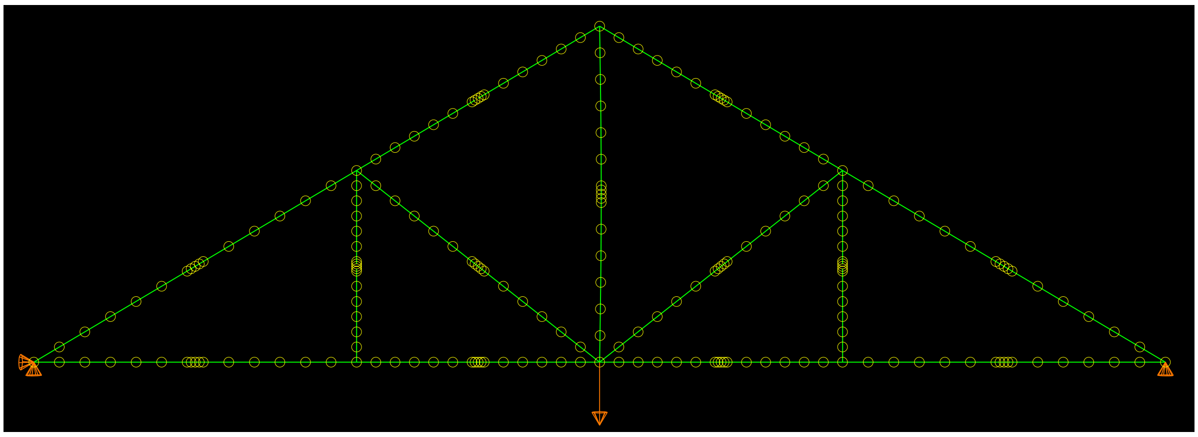


Figure 10.3: Example of a meshed Howrah structure in ABAQUS using B22 elements. Node labels were left out for clarity.

The Howrah structure is configured in such a way that differential loading is inevitable and will greatly be influenced by the geometrical parameters. The entire FEA structure is consequently meshed with B22 elements. With 13 truss members, the complexity of the FEA input file while including the possibility for material defects is extensive as shown in Figure 10.3 displaying an example of a meshed Howrah structure in ABAQUS. In contrast, the model meshed with only 13 NmT2 elements is quick to build. With regard to the analysis, the downward loading will induce a reaction force at the boundary in node 1. Note that unlike the previous cases, the displacement due to loading is not translated to $\Delta a_{e,\text{axial}}$ for the truss member at the boundary due to the multi-truss nature of the Howrah structure. Additionally, final displacement profiles are compared. This was introduced in scenario three of the previous case study. The downward loading on the Howrah structure will induce horizontal and vertical displacements in all nodes. However, only 8 of them can be compared between the NmT2 and B22 elements since the neuromorphic FEA model only requires 13 elements whereas the standard model requires well over 100.

10.1.1. Scenario 1: No defect seeding

In the first scenario, the Howrah structure is evaluated with no defect seeding (Figure 10.4). In terms of error analysis, the same error metrics which were presented before will remain in this section, and the displacement error $\bar{\epsilon}_{\text{disp}}$ will be computed relative to the standard FEA model and averaged over all 8 nodes. In addition to the usual error metrics, it is also indicated which truss member buckles first thereby causing a drop in overall structural integrity. The computing time between the model meshed with B22 elements and the model meshed with NmT2 elements is also compared. The error

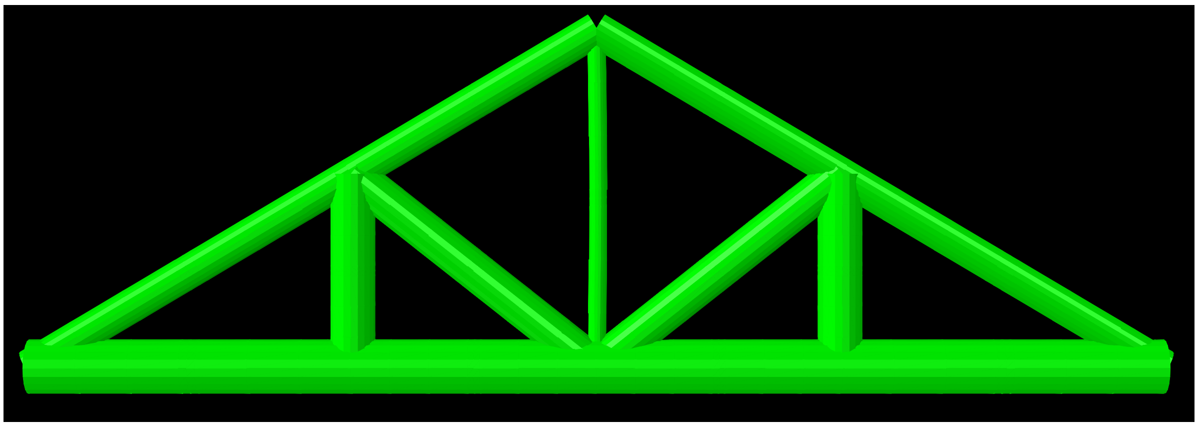


Figure 10.4: Example of a rendered Howrah structured in ABAQUS with no defect seeding.

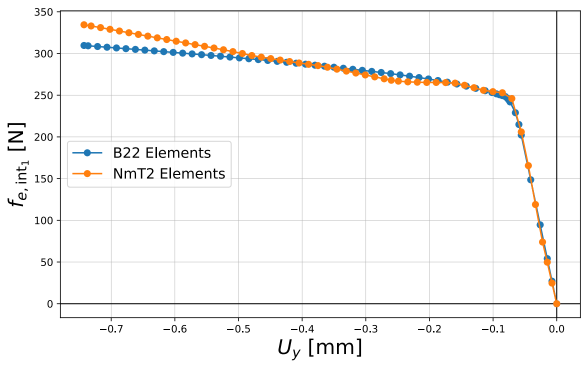
analysis for all 3 randomly generated models may be found in Table 10.1.

Although 3 randomly generated models is relatively few compared to the previous two case studies, it is still enough to display some variation in the failure scheme. For instance, the first member to buckle in model 1 is the member 2 which is a horizontal component when referring back to Figure 10.2. In contrast, in model 2, the first member to buckle is member 11. This confirms that the geometrical selection does influence the point where structural integrity is first lost. However, when referring to the load profiles in Figure 10.2, there is no sharp decrease into the post buckling regime which was seen in the previous two case studies. This is thanks to the integrity of the Howrah structure. In the three generated models, as soon as one truss member begins to buckle, it is supported by the remainder of the structure. Although there is a decrease in the structural integrity, there is no sharp decrease. In addition, the imposed displacements are relatively small. If the structure were pushed further, then the sharp decrease in structural integrity would occur. This, however, would be outside the modelling domain of the NmT2 element.

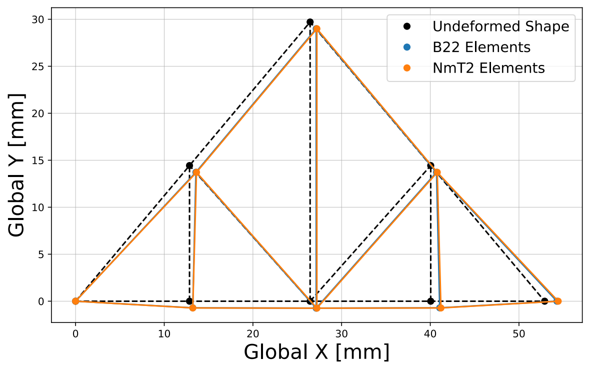
With respect to the error analysis, the overall performance of the NmT2 elements versus traditional B22 elements is exceptional. All R^2 values are greater than 0.97 indicating a high degree of correlation shown in Figures 10.5 through 10.7. The final displacement profiles are also well matched with average error deviations no greater than 0.1mm, all while decreasing the computing time by nearly an order of magnitude. However it is important to reiterate that no defects are included at this point; they will be introduced shortly. It is important to note that the displacement profiles do not capture the full deformation of the structure. Theoretically, the displacements of all the nodes in the mesh shown in Figure 10.3 can be extracted, which would illustrate which truss members buckle and how much deformation out of the axial plane is occurring. However, the NmT2 elements only possess nodes at the ends of each truss member and are unable to show the displacement of the axial plane for the entirety of the truss member. Therefore, only the displacements from the nodes at the ends of the truss members are compared between the NmT2 and B22 elements.

Table 10.1: Error and temporal analysis across three models with no defect seeding.

Model	$\epsilon_{\max}$ [%]	$\epsilon_{\min}$ [%]	$\bar{\epsilon}$ [%]	R^2 [-]	$\bar{\epsilon}_{\text{disp}}$ [mm]	t_{B22} [s]	t_{NmT2} [s]	First buckle
1	21.827	0.054	3.654	0.974	0.074	385.68	31.75	Member 2
2	6.183	0.028	3.906	0.978	0.016	378.12	30.11	Member 11
3	8.014	0.808	2.414	0.993	0.053	390.94	29.83	Member 1
Average	12.008	0.297	3.325	0.982	0.047	384.91	30.56	

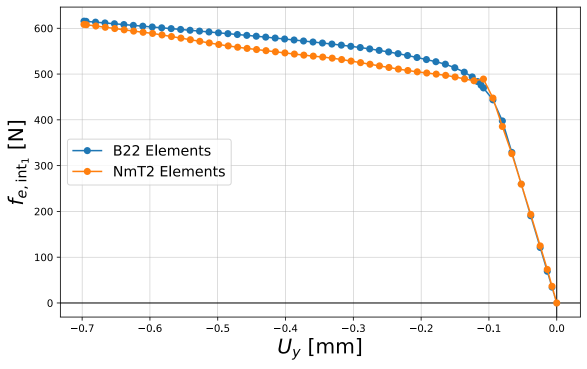


(a) Nodal reaction force at node 1.

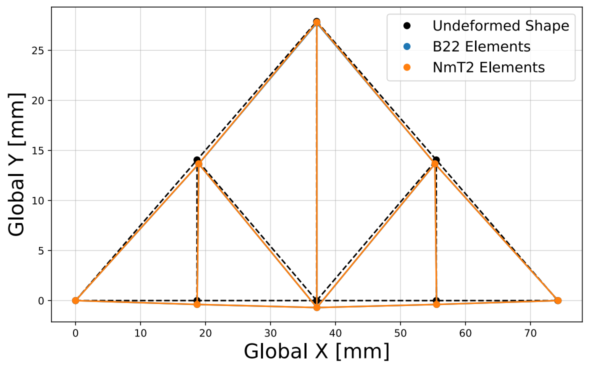


(b) Displacement profiles.

Figure 10.5: Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Geometrical parameters: $\{L_1, L_2, L_3\} = [12.85\text{mm}, 13.60\text{mm}, 29.71\text{mm}]$ and $\{r_1, r_8, r_9, r_{10}, r_{11}\} = [0.59\text{mm}, 1.01\text{mm}, 1.44\text{mm}, 0.86\text{mm}, 1.07\text{mm}]$. No defect seeding.

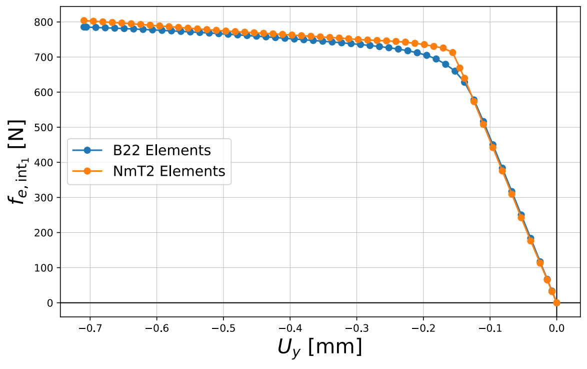


(a) Nodal reaction force at node 1.

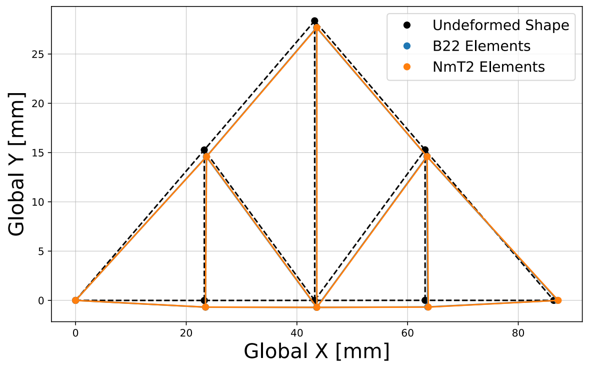


(b) Displacement profiles.

Figure 10.6: Comparison of converged results from model 2 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Geometrical parameters: $\{L_1, L_2, L_3\} = [18.69\text{mm}, 18.41\text{mm}, 27.89\text{mm}]$ and $\{r_1, r_8, r_9, r_{10}, r_{11}\} = [1.28\text{mm}, 1.47\text{mm}, 1.50\text{mm}, 0.98\text{mm}, 1.24\text{mm}]$. No defect seeding.



(a) Nodal reaction force at node 1.



(b) Displacement profiles.

Figure 10.7: Comparison of converged results from model 3 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Geometrical parameters: $\{L_1, L_2, L_3\} = [23.26\text{mm}, 19.95\text{mm}, 28.37\text{mm}]$ and $\{r_1, r_8, r_9, r_{10}, r_{11}\} = [1.33\text{mm}, 1.59\text{mm}, 1.42\text{mm}, 1.75\text{mm}, 1.77\text{mm}]$. No defect seeding.

10.1.2. Scenario 2: Including defect seeding

For the second scenario, random defect seeding is included to represent material defects. To draw yet again from a real-world example, the Howrah bridge has experienced considerable structural corrosion over the years caused by bird feces. This coupled with the constant high loads, are mainly responsible for the sagging of the bridge in some regions.

Embedding random defects in a conventional FEA model is no simple task. Partitions have to be defined and specified sections of reduced radius also have to be included. The process has not changed throughout any of the case studies. The only difference for the Howrah structure is that the defects have to be included in 13 structural elements. Building this model takes a considerable amount of time, and programming it through an automated input file process is no faster. The fact that the NmT2 elements have defect-modelling capabilities built into them is a major time-saving feature.

Figures 10.8 and 10.9 show an example of the final structural geometry while including material defects. As may be seen, the radial defects are randomly distributed throughout the structure and will greatly influence the point at which buckling of a truss member occurs. In this particular case, the presence of defects had the most effect on truss member 5 since it buckles first in Figure 10.9.

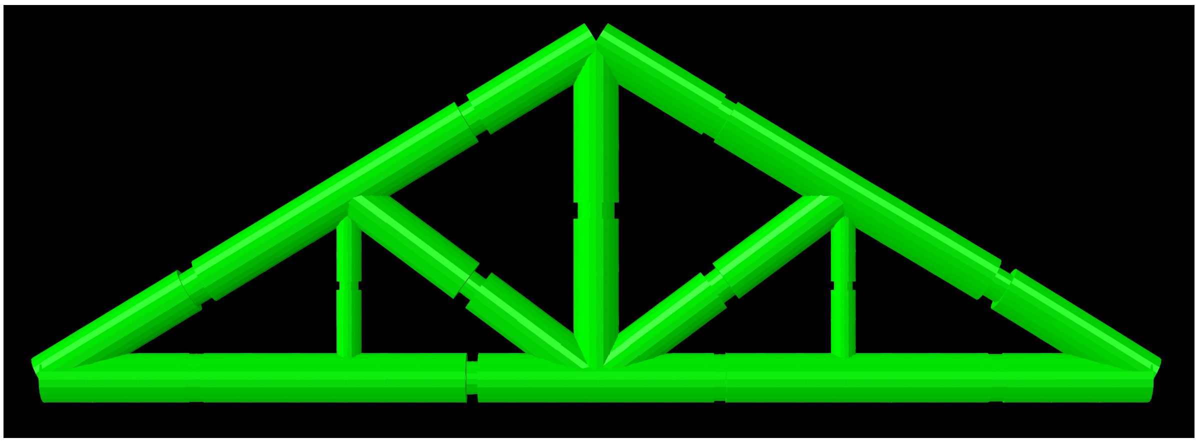


Figure 10.8: Example of a rendered Howrah structured in ABAQUS with defect seeding.

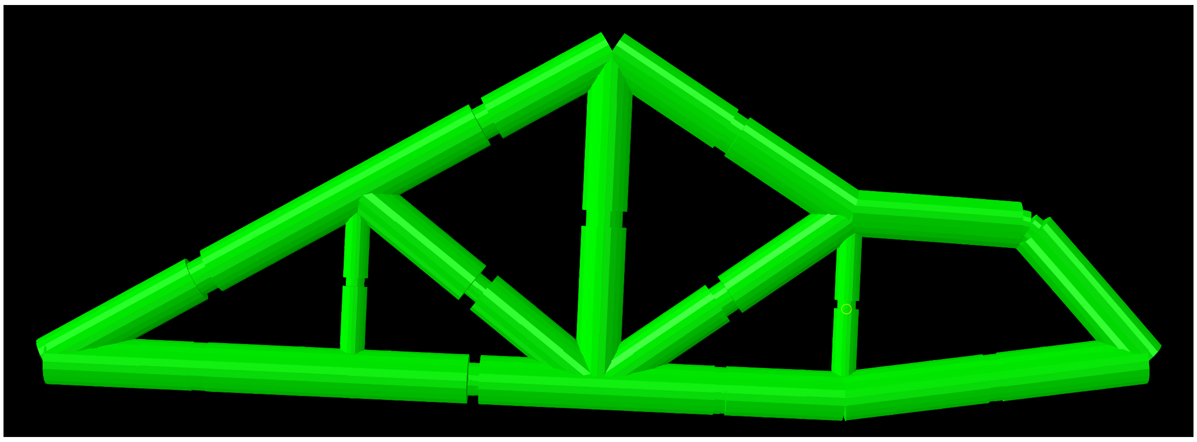


Figure 10.9: Example of a loaded Howrah structured in ABAQUS with defect seeding, which buckles in truss member 5 due to structural defects.

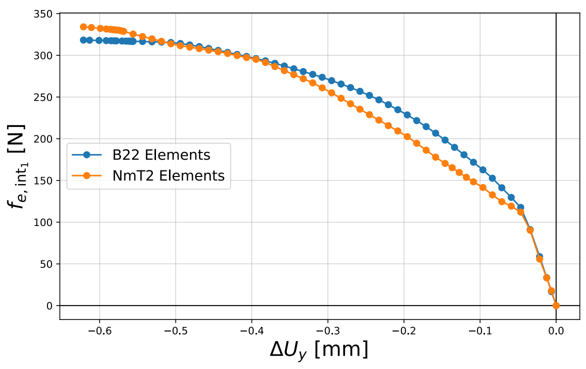
Just as before, 3 models were randomly generated, and the error analysis is shown in Table 10.2. Their corresponding load-displacement curves and global displacement profiles are shown in Figures 10.10 through 10.12. The overall errors have slightly increased as expected due to the increased difficulty of modelling structural deformation with defects, though remain closely correlated with the B22 elements. The presence of radial defects throughout the Howrah structure does compromise its structural integrity, which can be seen through the load-displacement curves. They show a higher degree of curvature indicating that certain structural members are starting to buckle. The most notable observation, however, concerns Figure 10.11.

Figure 10.11 corresponds to the Howrah structure shown in Figure 10.9. Here we notice a transition to the post-buckling regime that is more violent than the other cases, and similar to those seen in the past two case studies. In this particular case, the first member to buckle is truss member 5 due to the structural defect. There is no support from other truss members or from the boundary condition at node 5 (which is simply supported), to support the structural member as it buckles. The buckling of this truss member completely cripples the structural integrity, manifested by the sharp load drop in Figure 10.11. Despite this, the NmT2 elements manage to model the entire buckling effect and how it propagates throughout the structure, as well as match the final global displacements.

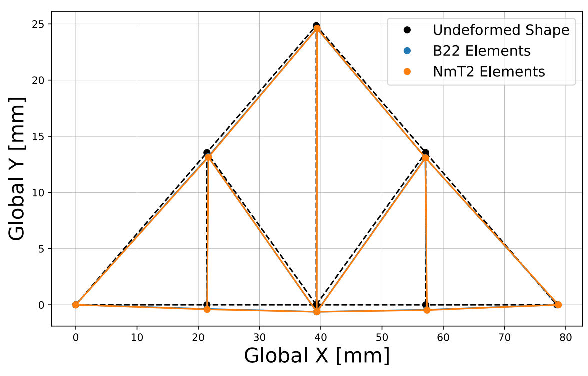
Over these three models, the NmT2 elements manage to maintain a high degree of correlation with the traditional B22 elements with an average $R^2 = 0.956$. The average absolute error, $\bar{\epsilon}$ is also maintained below 10%. The most exciting aspect is without a doubt the computing time. While maintaining a high degree of accuracy, the neuromorphic elements average roughly 30 seconds per model whereas the models meshed with B22 elements take more than seven minutes to converge.

Table 10.2: Error and temporal analysis across three models with random defect seeding.

Model	ϵ_{max} [%]	ϵ_{min} [%]	$\bar{\epsilon}$ [%]	R^2 [-]	$\bar{\epsilon}_{\text{disp}}$ [mm]	t_{B22} [s]	t_{NmT2} [s]	First buckle
1	14.929	0.301	5.624	0.967	0.037	415.27	30.95	Member 11
2	27.748	0.639	7.151	0.982	0.058	495.11	31.23	Member 5
3	30.088	0.421	7.857	0.919	0.035	403.04	33.07	Member 7
Average	24.255	0.453	6.883	0.956	0.043	437.81	31.75	

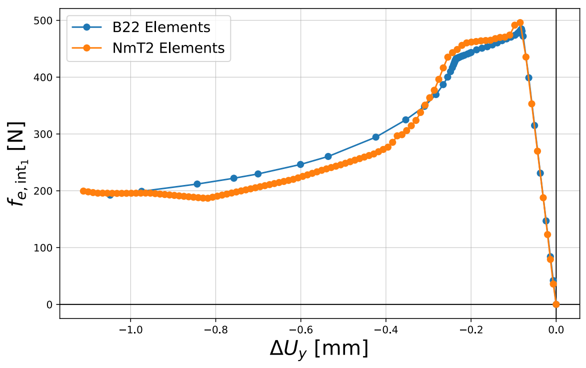


(a) Nodal reaction force at node 1.

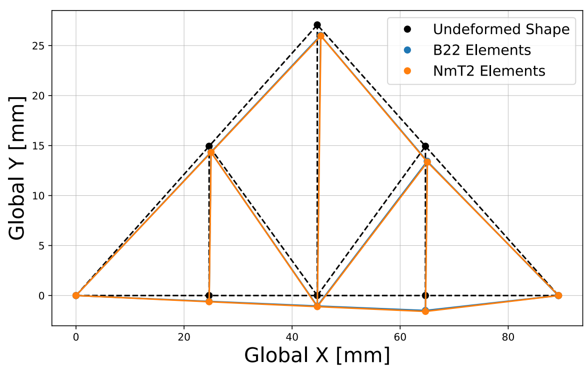


(b) Displacement profiles.

Figure 10.10: Comparison of converged results from model 1 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Geometrical parameters: $\{L_1, L_2, L_3\}$ = [21.41mm, 17.86mm, 24.85mm] and $\{r_1, r_8, r_9, r_{10}, r_{11}\}$ = [0.92mm, 1.42mm, 1.39mm, 0.57mm, 1.08mm]. Random defect seeding.

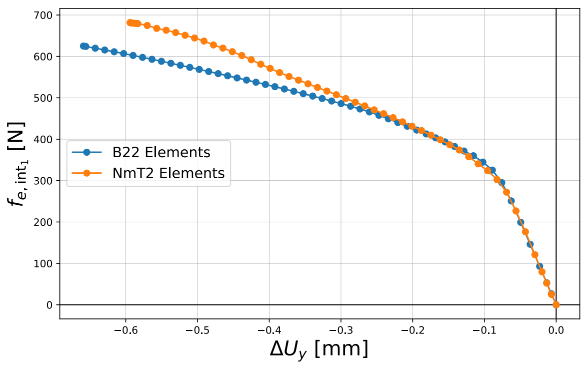


(a) Nodal reaction force at node 1.

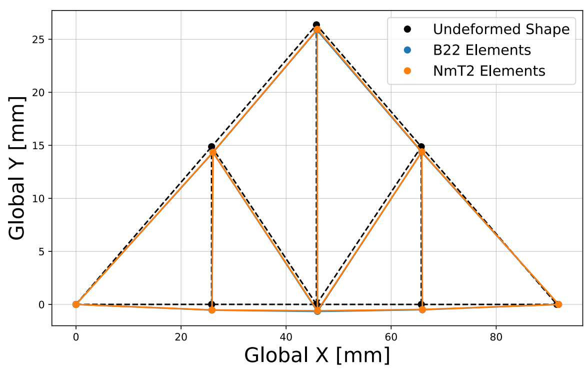


(b) Displacement profiles.

Figure 10.11: Comparison of converged results from model 2 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Geometrical parameters: $\{L_1, L_2, L_3\}$ = [24.64mm, 20.02mm, 27.07mm] and $\{r_1, r_8, r_9, r_{10}, r_{11}\}$ = [1.98mm, 1.87mm, 0.99mm, 1.77mm, 0.81mm]. Random defect seeding.



(a) Nodal reaction force at node 1.



(b) Displacement profiles.

Figure 10.12: Comparison of converged results from model 3 between traditional FEA elements (B22) and NmT2 elements due to a downward displacement U_y at node 3. Geometrical parameters: $\{L_1, L_2, L_3\}$ = [25.82mm, 19.96mm, 26.36mm] and $\{r_1, r_8, r_9, r_{10}, r_{11}\}$ = [1.46mm, 1.66mm, 1.78mm, 0.80mm, 1.72mm]. Random defect seeding.

10.2. Closure

The Howrah bridge proved to be an interesting inspiration for a multi-truss structure. The goal of this final case study was to push the capabilities of the neuromorphic element even further and put them in a scenario where differential loading was inevitable. Additionally, the structural complexity had to be high enough such that it is not possible to validate it by hand, which is immediately the case when considering the random radial defect seeding and buckling scenario. The most important observations throughout this final study are:

1. The NmT2 elements remain robust even when deployed on more complex multi-truss structures; and are able to accurately capture the nodal reaction forces regardless of the degree of nonlinearity of the deformation.
2. In addition to the nodal forces, the NmT2 elements also accurately capture the displacement profile of the structure.
3. The gain in the time required to build the FEA model with NmT2 elements is substantial when compared to the time it takes to construct a traditional model with B22 elements.
4. The NmT2 elements managed to reduce the overall computing time by more than 1,000%.

From a model development standpoint, the NmT2 elements are far easier to work with since an entire structural member can be directly replaced by a singular NmT2 element. This is not the case when dealing with traditional B22 elements where each structural member has to be partitioned and meshed to correctly introduce the radial defect, all while ensuring solution convergence. This can easily lead to FEA models with more than 150 B22 elements. Given that NmT2 elements already have defect modelling capabilities built-into them, they are far easier to work with. Case 2 similarly showed the ease of using the NmT2 elements compared to the traditional FEA elements. Although there, the effort to build the FEA model was much less than that required for Case 3, due to Case 3's greater number of structural members. Throughout all three case studies, the amount of time spent building the FEA models was not discussed since it depends on the user's experience. The time required to build the FEA model will be reconsidered in the final discussion.

For loading scenario 1, which omitted radial defect seeding, the results from the NmT2 elements were in perfect coherence with those using B22 elements, both in terms of boundary reaction force and the displacement profiles, and at a fraction of the computational effort. This was the same outcome for the scenario 2 with random defect seeding. In one model of scenario 2, a truss member on the edge of the structure was the first to buckle and did so abruptly, quickly reducing the overall structural integrity. The neuromorphic elements were capable of accurately modelling the structural behaviour, despite the nonlinearity of the deformation regime or the presence of defects. In addition, the NmT2 elements were substantially more efficient in terms of computing time, which is summarized in Table 10.3. In both scenarios, the NmT2 reduced the computing time by over 1,000%. This affirms the potential of data-driven elements in multi-truss structures and concludes the case study series of this thesis.

Table 10.3: Computational efficiency of traditional FEA elements versus 13 NmT2 elements for case study 3.

	Average computing time [s]		
Defect seeding	B22	NmT2	**Gain in computational efficiency** [%]
No defects	384.91	30.56	1,159.59
Random	437.81	31.75	1,278.93

11
Discussion

The purpose of this thesis is to investigate whether it is possible to embed a trained neural network into an active FEA simulation to reduce its computational expense without compromising solution accuracy. Computational expense is referred to as the amount of time required for an FEA model to converge to a solution. Fast computing hardware or cloud-based resources can boost CPU performance in a brute-force manner. However, these can quickly become very expensive, and they are not always readily available, especially when conducting offline FEA simulations from a personal computer for example. Therefore, alternate methods which improve computational efficiency while bypassing the need for elaborate computing systems are highly desirable.

The neuromorphic element (designated as NmT2) was developed as a custom finite element that can be easily imported within ABAQUS and used to model planar multi-truss structures. At the core of the subroutine, the NmT2 element has a trained neural network (called the *neuromorphic engine*), which computes the internal nodal forces at the element's extremities based on the nodal displacements and the truss member's geometry. This circumvents the traditional FEA method of computing the internal nodal force vector by using a data-driven framework. The neuromorphic engine itself was trained on over half a million datasets and accounted for factors such as material plasticity, a post-buckling regime, and the presence of simulated structural defects at the centre of the truss member. The approach accommodated the deformation regimes of beam elements (such as deformation out of the axial plane), as well as the constraints of truss elements (only axial loading/deformation is permitted).

The final discussion of this thesis is divided into two parts: the first part concerns the development of the neural network, and the second part focuses on the practical applications of the data-driven, NmT2 finite element.

11.1. The neuromorphic engine: Bypassing the trial-and-error approach

The creation of the neuromorphic engine was prompted by the desire to reduce the scope for human error during the development phase. A neural network's settings (or hyperparameters), coupled with the application and data at hand, define its modelling capabilities. Given there are no explicit boundaries on any of the hyperparameters, this creates a theoretically infinite number of neural network configurations. A trial-and-error approach is typically adopted to define the best combination of hyperparameters, however this means that only a very small number of neural architectures can be tested, and does not guarantee that the final choice is the best possible network.

It also ironically defeats the purpose of creating an intelligent computing framework by adopting an unintelligent approach. Unless the application at hand can be related to an example in literature, then a trial-and-error approach should not be used when building a neural network.

The neuromorphic engine does not use trial and error to build a network. It chooses its own setting without any human interference. The multi-objective optimization algorithm that is developed randomly samples the hyperparameter space, and tests the network on a randomly sampled portion of the overall dataset using Latin Hypercube sampling. The network is then trained for 100 epochs or less if the network's performance saturated. Once completed, the performance metrics are stored and another network is developed and tested. Once the algorithm reaches a state of diminishing returns indicated by 5 consecutive models showing no performance improvement, the optimization scheme is halted and the best configuration in the tested selection is saved and deployed on the entire data repository. In total, 5,824 network configurations are tested out of the 400,000 permutations in the hyperparameter design space, meaning that less than 1.5% were tested before determining that the algorithm had already selected an adequate configuration.

Table 11.1 summarizes the global hyperparameters of the selected network. Overall, the network architecture is relatively simple compared to the problem at hand. For such a complicated regression problem, a deeper neural network might have been expected. This shows that the multi-objective optimization approach does work and managed to produce an efficient network with a high-fidelity modelling capability manifested by the 0.999 R^2 metric. The total computational time required to determine a suitable architecture once the algorithm was initiated was roughly 42 hours. This is substantially less than the expected time it would have taken to find a suitable configuration using the trial-and-error approach.

Table 11.1: Final global hyperparameters of the neuromorphic engine.

Final network shape	Brick configuration
Number of hidden layers	5
Number of nodes per layer	100
Activation function in hidden layers	Rectified linear unit (ReLU)
Activation function in output layer	Hyperbolic tangent (Tanh)
Attainable R^2 metric in 500 epochs	0.9999

Once the final neural network was built, it was trained on the full data repository of over half a million datasets and managed to model the full deformation field of a truss member, which included material plasticity, post-buckling, and the presence of manufacturing defects in the structure. The success of the neuromorphic engine's modelling capabilities can be largely attributed to the data normalization process which segregated tensile and compressive displacement by flipping the normalization domain. This allowed the network to distinguish between axial tension and compression, and learn the deformation domain accordingly. Once finalized, the neuromorphic engine was built into the user-element subroutine and the NmT2 element could be deployed in an FEA setting.

11.2. Assessing the practical application of the NmT2 element

To assess the practical application of the NmT2 element, three case studies were defined and were constructed to incrementally test the capabilities of the neuromorphic element. The first case study represented a single horizontal truss member that was axially loaded with simply-supported boundary conditions (shown in Figure 11.1). This corresponded exactly to the type of model that was built during the data generation phase used to train the neuromorphic engine. The purpose of the first case study was to provide a first-order verification of the neuromorphic element. During the development of the neuromorphic engine, the network was able to model the data with correlation of $R^2 = 0.999$ across the board. The only difference between the first case study and the models that the neuromorphic engine was trained on, is that the geometry of the truss member in case study one is randomly selected, and therefore not present in the data repository. Some very small discrepancies were observable, but the NmT2 remained robust and accurately modelled the entire deformation domain of 30 randomly generated truss members when compared to models meshed with traditional FEA elements (T2D2 in the case of tension and B22 in the case of compression). In addition to the maintained accuracy, the neuromorphic element already showed improvements in computational efficiency: it required on average 23 seconds for a simulation to reach completion. In contrast, the models meshed with traditional FEA elements required 40 seconds on average in the case of tension and 85 seconds in the case of compression. It is also worth noting that the time required to build the model was much faster with the NmT2 element. This is because the entire truss member could be meshed with a single NmT2 element. The traditional FEA models, on the other hand, required far more time due to the possible presence of a radial defect and the need to ensure solution convergence (which was very important in the case of post-buckling). The performance gains both in terms of modelling speed and reduction in computational time without compromising solution accuracy confirmed the potential of the NmT2 element and prompted the development of a second case study.

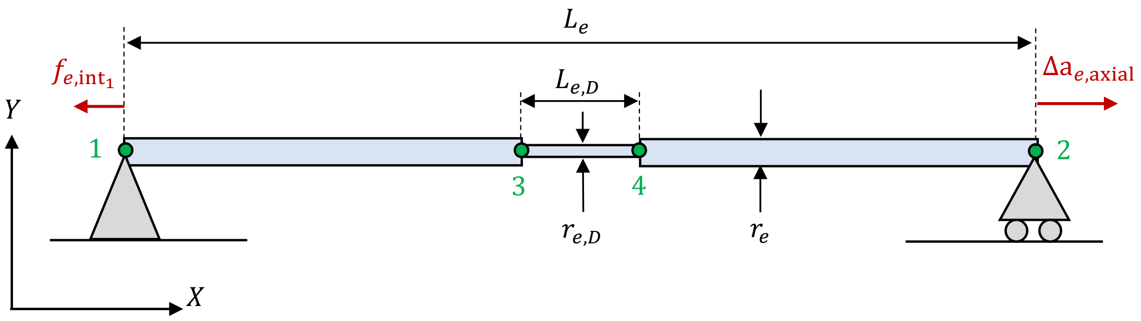


Figure 11.1: Case study 1: simply-supported beam with displacement loading (in this case, the structure is loaded in tension).

The second case study aimed to build on the first in an incremental fashion by introducing multi-element interaction. This lead to a structure consisting of two truss members, joined at the bottom forming a V-shape (shown in Figure 11.2). The joint between the two truss members was a pinned connection since the neuromorphic elements were not trained to model nodal rotations. Structural defects could again be included in both truss members. To ensure an incremental testing scheme relative to the first case study, it was assumed that both truss member possess the same geometrical parameters: the overall structure was therefore symmetric. From here, three loading scenarios were devised. The first scenario loaded the structure downward at the bottom node inducing tensile deformation in both members ($\Delta U_x = 0, \Delta U_y < 0$). The second scenario loaded the structure in the opposite direction (upward), such that both members would simultaneously buckle ($\Delta U_x = 0, \Delta U_y > 0$). It should be noted that having both members buckle at the exact same time is unrealistic, even if they both have the exact same geometry. In reality, one truss member would buckle before the other due to minor structural defects. The third and final scenario loaded the structure towards the right inducing differential deformation in both truss members ($\Delta U_x > 0, \Delta U_y = 0$). The left truss member would be in tension, whereas the right truss member would be in compression and would eventually buckle. A total of 30 configurations were ran-

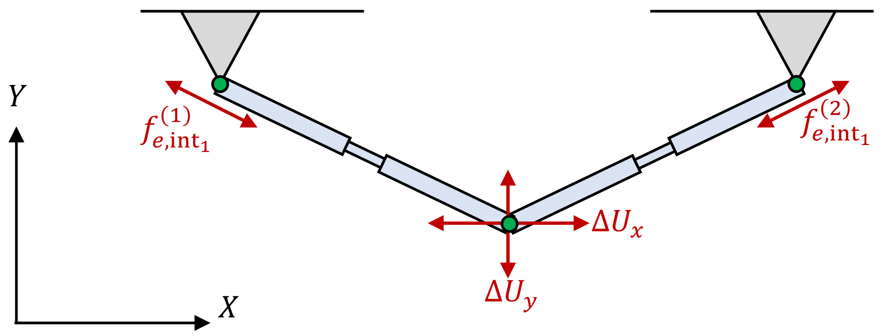


Figure 11.2: Case study 2: V-shaped structure of two truss members and loaded at its lower-most node.

domly generated for each loading scenario and the NmT2 elements showed an excellent coherence throughout all of them when compared to the models meshed with traditional FEA elements. This showed that the NmT2 elements were capable of multi-element interaction within an active FEA simulation, and maintained a low computational cost. The computational cost started to increase for the models meshed with traditional FEA elements. In addition, the difference in build time between the models meshed with neuromorphic elements and those meshed T2D2 or B22 elements was considerable. Because the NmT2 elements allow the user to directly replace an entire truss member with a single element, building the model can be done very quickly. The success of the second case study motivated the development of a third and final case study of even greater complexity.

The third case study was aimed to test the NmT2 element in a more complex multi-truss structure. The Howrah structure is a repeated unit from the Howrah bridge in Kolkata, India, and consists of 13 truss members in a triangular arrangement (shown in Figure 11.3). The structure is simply supported at its extremities and loaded downwards at the middle intersection point. The models were designed such that the radii and lengths of each truss-member were randomly generated, but constrained such that the structure would remain symmetric. The defects were then introduced. Two scenarios were considered: one with no structural defects and another with randomly seeded

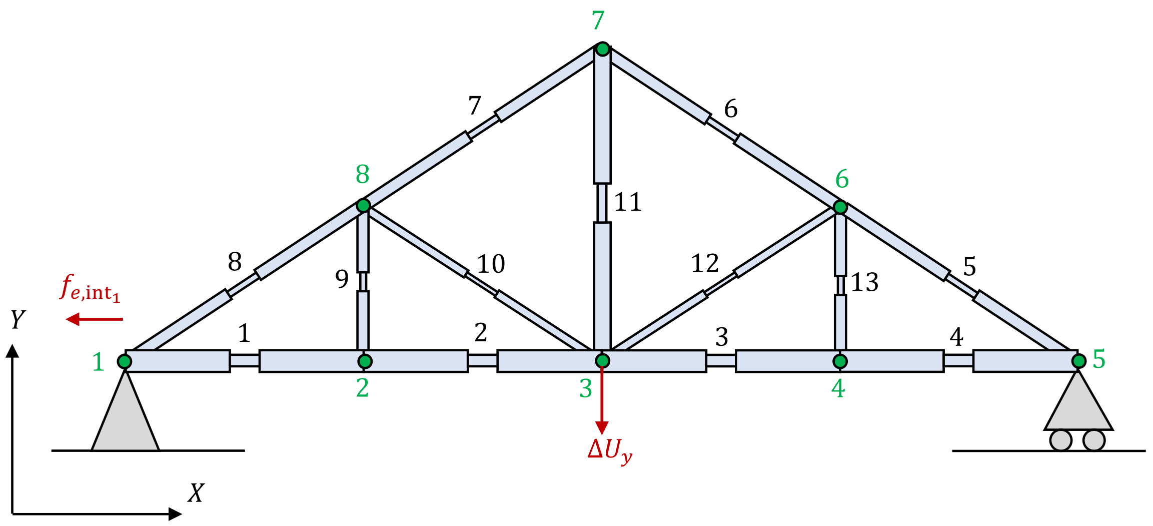


Figure 11.3: Case study 3: Howrah structure consisting of 13 truss members. The structure is pinned at node 1 and simply supported at node 5. A downwards displacement load (ΔU_y) is applied at node 3.

defects. This is intuitively a very complex structure since, regardless of the geometry of the truss members, some will be in tension and others in compression. Differential deformation is therefore unavoidable. Given that the geometries are randomly assigned, it is impossible to predict which truss members will buckle first. Case study three thus presents an opportune environment to more comprehensively test the capabilities of the NmT2 elements. Only three models were designed for each configuration since the modelling and computing times are substantially higher when using traditional FEA elements. The neuromorphic elements remained robust and modelled the load-displacement curves as well as the global displacement profiles of the Howrah structure with the same accuracy as in the previous two case studies. The increase in computational efficiency from the NmT2 elements is over an order of magnitude (> 1,000%) relative to the models meshed with traditional FEA elements.

Table 11.2: Overview of key metrics between all three case studies. Absolute errors and R^2 metrics refer to the amount of deviation between FEA models meshed with NmT2 elements relative to FEA models meshed with traditional elements (T2D2/B22). The gain in efficiency and loss in accuracy relative to FEA models meshed with traditional elements (rows in red) are averaged over all scenarios for a given case study.

Entity	**Case study 1** **Single member**	**Case study 2** **V-structure**	**Case study 3** **Howrah structure**
Number of truss members	1	2	13
Number of NmT2 elements	1	2	13
Number of FEA elements (T2B2 or B22)	= 3 T2D2 (Tension) ≈ 10 B22 (Compression)	= 6 T2D2 (Scenario 1) ≈ 20 B22 (Scenario 2) ≈ 20 B22 (Scenario 3)	≈ 175 B22 (No defects) ≈ 175 B22 (Defects)
Modelling time NmT2 elements	< 5 min	< 10 min	< 20 min
Modelling time FEA elements (T2D2/B22)	≈ 20 min	≈ 45 min	≈ 90 min
NmT2 increase in modelling efficiency	≈ 300 %	≈ 350 %	≈ 350 %
Computing time NmT2 elements	≈ 0.4 min (Tension) ≈ 0.4 min (Compression)	≈ 0.4 min (Scenario 1) ≈ 0.4 min (Scenario 2) ≈ 0.4 min (Scenario 3)	≈ 0.5 min (No defects) ≈ 0.5 min (Defects)
Computing time FEA elements (T2D2/B22)	≈ 0.7 min (Tension) ≈ 1.4 min (Compression)	≈ 0.8 min (Scenario 1) ≈ 1.8 min (Scenario 2) ≈ 2.4 min (Scenario 3)	≈ 6.4 min (No defects) ≈ 7.3 min (Defects)
NmT2 increase in computing efficiency	≈ 160 %	≈ 320 %	≈ 1,300 %
Average absolute error relative to FEA model	≈ 1.09 % (Tension) ≈ 6.05 % (Compression)	≈ 2.37 % (Scenario 1) ≈ 3.25% (Scenario 2) ≈ 3.72% (Scenario 3)	≈ 3.33 % (No defects) ≈ 6.88 % (Defects)
Average attainable R^2 relative to FEA model	≈ 0.99 (Tension) ≈ 0.95 (Compression)	≈ 0.99 (Scenario 1) ≈ 0.99 (Scenario 2) ≈ 0.99 (Scenario 3)	≈ 0.98 (No defects) ≈ 0.96 (Defects)
Loss in accuracy	≈ 4 %	≈ 3 %	≈ 5 %

The results from all three case studies confirm that the developed neuromorphic element is able to accurately model planar multi-truss structure while reducing the computing and modelling time, thereby improving the overall computational efficiency. Incorporating material plasticity, post-buckling deformation, and structural defects were no problem at all for the neuromorphic element. The average results for all three case studies are summarized in Table 11.2.

A final point of reflection concerns the chosen application which was designated as multi-truss structures resulting from biomimicry optimizers which could then be 3D-printed. For all three case studies, the nodes at the extremities of the truss members were pinned or simply supported, thereby allowing rotational degrees of freedom. This is self-explanatory for the first case study since only a single truss member was considered. For the second and third case studies, caution had to be exercised when meshing the structure with beam (B22) elements. Beam elements possess rotational degrees of freedom. This allows them to bend or and deform out of the axial plane, which makes them suited for post-buckling analyses. If the entire structure were to be designed and meshed as a whole within ABAQUS, then the joints between truss members would no longer be pinned. The joined truss members would share their rotational degrees of freedom with one another, which does not correspond to a pinned joint for which the NmT2 element was trained. Therefore, when meshing a multi-truss structure with B22 elements, the rotational degrees of freedom at the joints had to be disabled such that the joints correspond to pinned connections.

A multi-truss structure with pinned connections does not correspond to a 3D-printed structure at this stage of development. 3D-printed structures are produced as a single product, therefore, the multitude of truss members within the product are not joined with pinned connections, but are instead fused together. With this in mind, the neuromorphic element is not well suited for such 3D-printed structures. In order to adapt it for 3D-printed organic structure, the user would have to incorporate the shared rotational degrees of freedom at the joints between truss members. This will be discussed in the following chapter. For now, the NmT2 element is only suited to model multi-truss structures with pinned connections between truss members. Rather than seeing this as a limitation in that the NmT2 does not accurately represent the deformation of 3D-printed organic structures, it should be considered an opportunity opening up new applications such as modelling human bone joints.

11.3. Final summary: Addressing the initial research questions

At the beginning of the thesis, a series of research questions were formulated to help guide the overall project and break down the guiding research objective. Now is the time to address these questions.

RQ1. Is it possible to develop an intelligent framework for neural network design such that the user does not need to manually select and test all parameters to create a high-fidelity model?

Yes it is possible. The multi-objective algorithm created to develop the neuromorphic engine essentially removes the human element from the development phase of a neural network. As a result, the intelligent framework decides on the best set of hyperparameters and builds the final network entirely on its own. This is achieved by building a series of neural networks and testing them on a randomly sampled portion of the overall data. The algorithm does not test every network configuration, since this would be just as inefficient if not moreso as the traditional trial-and-error approach. Instead, only a series of networks are tested within the hyperparameter design space. Once the algorithm reaches a state of diminishing returns in terms of observed network performance, it then terminates assuming that a satisfactory configuration has been found. This results in less than 20%

of the total configurations being tested, providing the end user with an efficient and high-fidelity neural network.

RQ2. Is it feasible to embed a trained neural network into an active finite element analysis? Furthermore, can the neural network-based elements work together without disrupting the rest of the FEA environment?

Absolutely. The neuromorphic engine was embedded in the user-subroutine of a custom finite element to compute the internal nodal force vector as a function of the nodal displacements and truss member's geometry. When deployed in ABAQUS, it was shown that an entire multi-truss structure could be meshed with multiple neuromorphic elements, and the active FEA simulation proceeded as per usual, undisrupted.

RQ3. Can a neural network-based finite element accurately capture highly nonlinear phenomena such as plasticity and post-buckling that would otherwise require complex FEA models?

Yes it can, provided that sufficient training data are available to capture the nonlinear phenomena. A single NmT2 element is capable for capturing the post-buckling deformation domain and includes material plasticity. Additionally, the capabilities were pushed even further by allowing the user to include structural defects in the truss member. This introduces a significant amount of nonlinear phenomena, which the NmT2 element was able to accurately capture. Specifically, nonlinear phenomena such as plasticity and post-buckling regimes were addressed through training the NmT2 element to distinguish between tension and compression. The most attractive part is that the neuromorphic element is designed to replace a single truss member in a multi-truss structure. If post-buckling deformation were to be captured using traditional FEA elements, then multiple beam elements (B22 for example) would have to be used for a single truss member, resulting in more complex mesh discretization schemes. Also, including a structural defect is not trivial in an FEA model. The neuromorphic element circumvents this complexity entirely, providing a simple element that can easily be "plugged in". Additionally, the data-driven element does not require expensive computing equipment and can be used in an offline setting: even from a personal laptop.

RQ4. If the previous question is answered positively, does the change in computational expense justify the development of complex neural network-based systems for FEA modelling?

This is an interesting question since the development cost of the neuromorphic element has not been discussed up until now. Building a data-driven element is not a trivial endeavour and the time required to develop it is substantial. Table 11.3 summarizes the various phases in the development of a data-driven element and shows my estimate of their respective cost in terms of number of weeks, assuming that no learning curve is required (which was not the case for this thesis). This means that the engineer developing the new tool is accustomed to machine learning programming, clearly understands the intricacies of FEA user-subroutines and where the neural network can be embedded within the user-subroutine. It also assumed that the user is well-versed in FORTRAN. The result is roughly an eight week (or two month) development cost in terms of required time: building the data repository and the neuromorphic engine are the most expensive tasks. Once deployed, however, the NmT2 element accelerates computational efficiency both in terms of the computing time for an FEA simulation, and the time required to build the FEA model (shown in Table 11.2). Building an FEA model with NmT2 elements is very fast and usually takes less than 20 minutes, regardless of the complexity of the multi-truss model. Using traditional elements, this takes far longer when building the model through the GUI and even longer if only an input file is built. Although this depends on the user's proficiency with the software, it would be wise to budget at least

Table 11.3: Estimated development cost of the NmT2 element assuming no learning curve is required.

Task	**Estimated time required [weeks]**
Defining the parameter design space	1
Building the data repository	3
Developing the neuromorphic engine	2
Interfacing the neural network in a user-element subroutine	1
Testing & verifying the NmT2 element	1
Total	8

90 minutes for developing the FEA model. Once the model is submitted for analysis, the neuromorphic elements always converge to a solution in less than 40 seconds, whereas the computing time of the model using traditional FEA elements can quickly become very long: up to 10 minutes for complex planar multi-truss structures. Although the computational efficiency and thus time spent will depend on the hardware of the user's computer and its interactions with cloud-based input/output, the neural network is expected to retain a relative advantage over traditional FEA methods.

If a data-driven approach is not adopted, then the initial development time could be spent building and running a traditional FEA model. Eight weeks is sufficient to build and run such a model for the desired application, and even iterate it a few times. Despite the computational gains of the neuromorphic element when deployed, the development cost is substantial and could be argued to offset any computing gains when deployed. However, the true strength of the neuromorphic element lies in its flexibility, which cannot be matched by any traditional FEA elements. What this means is the NmT2 element is built to adapt to a wide range of geometrical configurations, and allows manufacturing defects to be included. It also allows for very simple mesh discretization schemes. When combining these aspects of the NmT2 element with the accelerated computing time, it is a powerful tool in any engineering optimization problem, especially when working in teams where the structural conditions or boundaries of design projects might change very quickly. In such cases, the models can be easily adapted and a new set of results can be obtained on the spot. This cannot be achieved with traditional FEA elements.

Therefore, because of the flexibility provided by data-driven elements and gains in computational efficiency, their additional cost in development time should not discourage their development. They should be seen as an extra tool an engineering group can use to tackle complex problems, as opposed to a replacement for traditional FEA methods. The above research questions were established to guide the driving research objective which is reiterated below:

The objective within the time-span of the thesis is to reduce the computational expense of a finite element analysis – without compromising its accuracy – by embedding neural networks trained on an element level into an active mesh.

The research has shown that the computational expense of a finite element analysis can be reduced using a neural network embedded at the element level. A new type of element with an embedded neural network was built: the neuromorphic element. This new element reduces the computing time and modelling time of multi-truss FEA problems while also reducing potential human error. Moreover, the introduction of a neuromorphic element provides reliability comparable to traditional FEA models, using standard statistical tests. These results affirm the potential of neural networks within active FEA simulations in the field of data-driven computational mechanics as a means to define complex nonlinear element formulations. The subsequent chapter focuses on potential future work that builds on the contents of this thesis.

12 Future Research

The current work showed that neural networks could be embedded into an active FEA simulation at the element level. The data-driven element managed to reduce computing time when analyzing multi-truss structures without any significant deviations in the final results, thereby providing the confidence and foundation to pursue the development of neuromorphic elements. This chapter contains a selection of potential future work which builds on the contents of this thesis. The work is divided into two categories: that pertinent to the development of the neuromorphic engine and by extension, neural networks in general; and that related to the usage of data-driven finite elements.

12.1. Neural network development

1. **Reduce amount of required training data**: With over a half a million datasets, the amount of data generated to train the neuromorphic engine was significant and time consuming. The data-generation phase in a machine learning project should never be underestimated, and it is where the most amount of time can be gained (or lost). Investigating the minimum amount of data required to train a neural network for finite element applications would be very useful for future work since it would reduce the number of FEA simulations required to build the data-repository. It would also help guide data gathering efforts where real-life conditions could provide additional information.

2. **Simplify network optimization**: The one aspect which the multi-objective optimization algorithm fails to consider in the development of the neuromorphic engine is the importance of training time (number of epochs) versus the hyperparameter selection. Chapter 2 mentions that a neural network with two hidden layers, each containing less than 50 neurons is sufficient to model the most complex regression problems. Testing this theory would simplify the design of the neural network considerably since no optimization algorithms or intensive trial-and-error approaches would be required. Only the activation functions, optimizer, loss function, and number of epochs the networks are trained for would have to be considered. This would greatly reduce the amount of time spent on the development of the neural network making it more efficient than other configurations, provided the final accuracy can be maintained.

3. **Empirical data generation**: Network training and element development would both benefit from datasets to help train networks or validate results. Monitoring of real-time information in a dynamic setting is becoming more routine in experimental and commercial aerospace settings (for example: aircraft engine performance monitoring, structural monitoring in launch

systems, etc.) As such monitoring is costly, AI developers should be working in concert with such monitoring efforts to ensure data acquisition is done cost efficiently.

12.2. Neuromorphic element development

1. **Scaling-up to 3D space**: Given that the current thesis focuses on 2D spatial applications of the neuromorphic element, a logical extension would be to build up the NmT2 element to 3D space (*NmT3*). This could be achieved with minimal effort since the data generation process and neuromorphic engine would remain exactly the same. Only the NmT2 user-element subroutine would have to be altered to account for 3D applications. The transformation matrix, force and position vectors would simply have to be slightly changed to account for the additional dimension. Scaling-up to 3D space would change the neuromorphic element's designation to *NmT3* and would permit a data-driven analysis of more complex multi-truss structures.

2. **Adding rotational degrees-of-freedom**: The NmT2 element was not considered suitable to model organic 3D-printed multi-truss structures since the element assumes that truss members joined via pinned connections. One way of overcoming this is to incorporate the rotational degrees-of-freedom in the neuromorphic element. This would require an entire new suite of FEA models for the element to learn how truss-members interact at rigid joints.

3. **Dynamic behaviour**: The NmT2 element could be applied to dynamic structural problems and compared to current FEA methods. Dynamic scaling is typically regarded as more complex than scaling to add a physical dimension using conventional modelling techniques. It may be that an NmT2 element could achieve similar efficiency improvements in a dynamic environment as it has demonstrated in a static environment.

4. **Alternate materials**: The NmT2 was trained on isotropic steel material data. Given the wide usage of composite structures in the aerospace industry, it would be a logical next step to attempt to apply the NmT2 element to such materials. If this is pushed to the extreme, the material parameter can also be extended as a user input into the neuromorphic element. This would mean that the NmT2 could adapt to steel, titanium, aluminum or composite materials. Although possible, it should be noted that the training domain to build a neural network that can adapt to multiple materials would be very large.

5. **Shell elements**: The truss represents one of the simpler FEA element configurations, however, the entire development process could also be applied to elements with higher degrees of freedom such as shell elements. The increase in degree-of-freedom would however, drastically increase the parameter design space of the neural network, making it more difficult to define the properties of the neuromorphic element.

13 Closing Remarks

Throughout this thesis and literature review, it has become apparent that neural networks applied to structural engineering are not taking advantage of all the available advancements in machine learning. The reason for this stems from a gap between the fields of computer science and engineering. Computer scientists continue to publish new advancements in neural network design, but their findings appear to be only applied to Big Data. This notion was reinforced during the AI & Big Data Expo 2019 in Amsterdam where out of the hundreds of companies present, only one was an engineering firm which specialized in designing machine learning algorithms to monitor anomaly occurrence in production lines.

Given the stringent safety and manufacturing tolerances in engineering, engineers are more reluctant to implement machine learning models that are essentially black boxes which they do not fully understand. The result is a disjointed state of research between the fields of machine learning and engineering where engineers are roughly 20 years behind when it comes to designing a neural network. Interestingly enough, engineering is not the sole industry averse to data-driven systems. The medical industry especially within the context of surgery and patient diagnosis also tends to resist the use of intelligent systems for similar reasons that engineers do.

The best way to mend this gap is to encourage collaborative research between practical and theoretical fields, which extends to industry. The problem should also be addressed from an educational standpoint: machine learning should not be exclusively taught to computer scientists, but to everyone. The modeling capabilities of machine learning algorithms have limitless potential across all fields ranging from architectural design, to engineering, and to marketing among others. Given that society is increasingly becoming data-driven it is only logical that this should be reflected in the education sector.

Although it might be a strange notion to grasp, data as a commodity is now considered very valuable. Wasting and neglecting data will be equated to wasting water before the end of the decade. Machine learning systems and intelligent algorithms possess computing potential which many choose to dismiss or undermine. However, if subjected to an appropriate course of training and properly formatted data, then such systems will be able to model the most volatile, nonlinear, and perhaps even chaotic phenomena in the world, including within society. For this to happen, researchers, engineers, and those in a position of developing data-driven systems, must embrace its potential and never lose sight of the value of data coupled with a continuous learning process. This is a marathon, not a sprint, and we all need to be in it together over the long term.

Bibliography

[1] *PAMCRASH Users Manuals.* ESI/PSI Inc. 2000, Version 2000.

[2] 3DEO. Intro to metal 3d printing processes - powder bed fusion. 2018. URL `https://news.3deo.co/metal-3d-printing-processes-powder-bed-fusion-dmls-sls-slm-lmf-dmp-ebm`.

[3] M. Abambres, K. Rajana, K. D. Tsavdaridis, and T. P. Ribeiro. Neural network-based formula for the buckling load prediction of i-section cellular steel beams. *Computers*, Vol. 8, 2018.

[4] Dassault Systemes ABAQUS. Defining plasticity in abaqus. URL `https://abaqus-docs.mit.edu/2017/English/SIMACAEGSARefMap/simagsa-c-matdefining.htm`.

[5] C. Aggarwal. *Neural Networks and Deep Learning.* Springer, 2018.

[6] J. Ahire. The artificial neural networks handbook: Part 1, 2018. URL `https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4`.

[7] J. Ahire. Ensemble methods: bagging, boosting and stacking. understanding the key concepts of ensemble learning., 2019. URL `https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205`.

[8] M. Aziz and Y. Sherif. Biomimicry as an approach for bio-inspired structure with the aid of computation. *Alexandria Engineering Journal*, Vol. 55:pp. 707–714, 2016.

[9] L. Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, Vol. 44:pp. 525–536, 1998.

[10] R. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[11] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. *NIPS Conference*, 2017.

[12] J. Bergstra and Y. Bengio. Random search for hyperparameter optimization. *Journal of Machine Learning Research*, Vol. 13:pp. 281–305, 2012.

[13] J. Bergstra, R. Dardenet, Y. Bengio, and B. Kégl. Algorithms for hyperparameter optimization. *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*, 2011.

[14] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28, 2013.

[15] M. A. Bessa, R. Bostanabad, Z. Liu, Z. 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.

[16] A. Bhande. What is underfitting and overfitting in machine learning and how to deal with it., 2018. URL `https://medium.com/greyatom/`.

[17] C. Bisagni and L. Lanzi. Post-buckling optimization of composite panels using neural networks. *Composite Structures*, Vol. 58:pp. 237–247, 2002.

[18] C. M. Bishop. Training with noise is equivalent to tikonov regularization. *Neural Computation 7*, Vol. 1:pp. 108–116, 1995.

[19] R. Borst, M. Crisfield, J. Remmers, and C. Verhoosel. *Non-linear Finite Element Analysis of Solids and Structures, Second Edition.* Wiley series in computational mechanics, 2012.

[20] L. Breimann. Bagging predictors. *Machine Learning*, Vol. 24:pp. 123–140, 1996.

[21] A. Bryson. A gradient method for optimizing multi-stage allocation processes. *Harvard University Symposium on Digital Computers and their Applications*, 1961.

[22] P. Bühlmann and B. Yu. Analyzing bagging. *The Annals of Statistics*, Vol. 30:pp. 927–961, 2002.

[23] D. Chakraborty. Artificial neural network based delamination prediction in laminated composites. *Materials and Design*, Vol. 26:pp. 1–7, 2005.

[24] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. *Proceedings of the 2017 SIAM International Conference on Data Mining*, 2017.

[25] Y. Chen and M. J. Zaki. Kate: K-competitive autoencoder for text. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages pp. 85–94, 2017.

[26] F. Chinesta, P. Ladeveze, R. Ibanez, J. V. Aguado, E. Abisset-Chavanne, and E. Cueto. Data-driven computational plasticity. *Procedia Engineering*, Vol. 207:pp. 209–214, 2017.

[27] A. Coates, H. Lee, and A. Ng. An analysis of single-layer networks in unsupervised feature learning. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 15 of JMLR: WCP 15:pp. 215–223, 2011.

[28] D. A. Coley. *An introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing, 1999.

[29] D. Cross. A new artificial leaf could help us combat climate change, 2019. URL `https://www.sustainability-times.com/environmental-protection/a-new-artificial-leaf-could-help-us-combat-climate-change/`.

[30] Multipysics Cyclopedia. Finite element mesh refinement, 2016. URL `https://www.comsol.com/multiphysics/mesh-refinement`.

[31] I. Czogiel, K. Luebke, and C. Weihs. Response surface methodology for optimizing hyperparameters. *Universität Dortmund, Fachbereich Statistik*, 2005.

[32] J. Dennis and J. Moré. Quasi-newton methods, motivation and theory. *Society for Industrial and Applied Mathematics.*, Vol. 19:pp.46–89, 1977.

[33] M. R. P. Elenchezhian, A. Nandini, V. Vadlamudi, R. Raihan, and K. Reifsnider. Detection and prediction of defects in composite materials using di-electric characterization and neural networks. *SAMPE Conference Proceedings*, 2018.

[34] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, and P. Vincent. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, Vol. 11:pp. 625–660, 2010.

[35] D. Fernández-Fdz and R. Zaera. A new tool based on artificial neural networks for the design of lightweight ceramic-metal armour against high-velocity impact of solids. *International Journal of Solids and Structures*, Vol. 45:pp. 6369–6383, 2008.

[36] F. Fritzen and O. Kunc. Two-stage data-driven homogenization for nmiscar solids using a reduced order model. *European Journal of Mechanics*, Vol. 69:pp. 201–220, 2018.

[37] 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 and Design*, Vol. 31:pp. 267–277, 2010.

[38] J. Ghaboussi, J. H. Garrett, and X. Wu. Knowledge-based modeling of material behavior with neural networks. *Journal of Engineering Mechanics*, Vol. 117:pp. 132–151, 1991.

[39] J. Ghaboussi, D. A. Pecknold, M. Zhang, and R. M. Haj-Ali. Autoprogressive training of neural network constitutive models. *Mechanical Engineering*, Vol. 42:pp. 105–126, 1998.

[40] J. Ghaboussi, X. Wu, and G. Kaklauskas. Neural network material modelling. *STATYBA - Civil Engineering*, Vol. 4:pp. 250–257, 1999.

[41] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28, 2013.

[42] T. H. E. Gulikers. An integrated machine learning and finite element analysis framework, applied to composite substructures including damage. *Master of Science Thesis, TU Delft*, 2018.

[43] M.T. Hagan and M.B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, Vol. 5:pp. 989–993, 1994.

[44] P. Hajela and L. Berke. Neural networks in structural analysis and design: An overview. *Computing systems in engineering*, Vol. 3:pp. 525–538, 1992.

[45] R. Hambli and N. Hattab. Application of neural network and finite-element method for multiscale prediction of bone fatigue crack growth in cancellous bone. 2013.

[46] Y. M. A. 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:pp. 989–1005, 2004.

[47] 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:pp. 35–50, 2009.

[48] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS 1993)*, Vol. 2:pp. 263–270, 1993.

[49] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Second Edition.* Springer, 2008.

[50] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations.* CRC Press, 2015.

[51] H. Hein and L. Feklistova. Computationally efficient delamination detection in composite beams using haar wavelets. *Mechanical Systems and Signal Processing*, Vol. 25:pp. 2257–2270, 2011.

[52] G. E. Hinton. A practical guide to training restricted boltzmann machines. *Technical Report 2010-003, University of Toronto*, 2010.

[53] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, Vol. 313:pp. 504–507, 2006.

[54] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation 2006*, 2006.

[55] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing coadaptation of feature detectors. *Department of Computer Science, University of Toronto*, 2012.

[56] R. Iati, M. Mizen, K. McPartland, and L. Tabb. *High Frequency Trading Technology: A TABB Anthology*. TABB Group, 2009.

[57] A. A. Javadi, T. Tan, and M. Zhang. Neural networks for constitutive modelling in finite element analysis. *Inverse problems in science and engineering*, 2003.

[58] K. Jensen. Performing topology optimization with the density method, 2019. URL `https://www.comsol.com/blogs/performing-topology-optimization-with-the-density-method/`.

[59] G. Kaklauskas, J. Ghaboussi, and X. Wu. Neural network modelling of stress-strain relationships for tensile concrete in flexure. *STATYBA - Civil Engineering*, Vol. 5:pp. 295–301, 1999.

[60] H. J. Kelley. Gradient theory of optimal flight paths. *Ars Journal*, Vol. 30:pp. 947–954, 1960.

[61] D. P. Kigma and J. L. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[62] T. Kirchdoerfer and M. Ortiz. Data-driven computational mechanics. *Computer methods in applied mechanics and engineering*, Vol. 304:pp. 81–101, 2016.

[63] R. Kohavi and D. H. Wolpert. Bias plus variance decomposition for zero-one loss functions. *Proceedings of the 13th International Conference on Machine Learning*, pages pp. 275–283, 1996.

[64] E. B. Kong and T. G. Dietterich. Error-correcting output coding corrects bias and variance. *Proceedings of the 12th International Conference on Machine Learning*, pages pp. 313–321, 1995.

[65] O. Kononenko and I. Kononenko. Machine learning and finite element method for physical systems modeling. *CoRR*, 2018.

[66] J. N. Kudva, N. Munir, and P. W. Tan. Damage detection in smart structures using neural networks and finite element analyses. *Smart Material Structures*, Vol. 1:pp. 108–112, 1992.

[67] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics.*, Vol. 22:pp. 79–86, 1951.

[68] H. B. Kupfer, H. D. Hilsdorf, and H. Rusch. Behavior of concrete under biaxial stresses. *ACI Journal*, pages pp. 656–666, 1969.

[69] P. Laboussière and N. Turkkan. Failure prediction of fibre-reinforced materials with neural networks. *Journal of Reinforced Plastics and Composites*, Vol. 12:pp. 1270–2180, 1993.

[70] L. Lanzi, C. Bisagni, and S. Ricci. Neural network systems to reproduce crash behavior of structural components. *Computers & Structures*, Vol. 82:pp. 193–108, 2004.

[71] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors on variation. *Proceedings of the 24th International Conference on Machine Learning*, 2007.

[72] H. Le, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area v2. *NIPS'07 Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages pp. 873–880, 2007.

[73] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng. On optimization methods for deep learning. *Proceedings of the 28th Internaitonal Conference on Machine Learning*, 2011.

[74] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2011.

[75] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems (NIPS 1989)*, Vol. 2, 1989.

[76] Y. LeCun, L. Bottou, G. Orr, and K. Muller. *Efficient backprop.* Springer, 1998b.

[77] C. S. Lee, W. Hwang, H. C. Park, and K. S. Han. Failure of carbon/epoxy composite tubes under combined axial and torsional loading: Experimental results and prediction of biaxial strength by the use of neural networks. *Composites Science and Technology*, Vol. 59:pp. 1779–1788, 1999.

[78] J. J. lee, J. W. Lee, J. H. Yi, C. B. Yun, and H. Y. Jung. Neural networks-based damage detection for bridges considering errors in baseline finite element models. *Journal of Sound and Vibration*, Vol. 280:pp. 555–578, 2005.

[79] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, Vol. 2:pp. 164–168, 1994.

[80] S. Liao, H. Kabir, Y. Cao, J. Xu, Q. Zhang, and J Ma. Neural-network modeling for 3d substructures based on spatial em-field coupling in finite element method. *IEEE Transactions on Microwave Theory and Techniques*, Vol. 59:pp. 21–38, 2011.

[81] P. F. Liu and J. Y. Zheng. Recent developments on damage modeling and finite element analysis for composite laminates: a review. *Materials & Design*, Vol. 31:pp. 3825–3834, 2010.

[82] Z. Liu, M. A. 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. 206:pp. 319–341, 2016.

[83] 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:pp. 1603–1613, 2010.

[84] M. H. Malik and A. F. M. Arif. Ann prediction model for composite plates against low velocity impact loads using finite element analysis. *Composite Structures*, Vol. 101:pp. 290–300, 2013.

[85] H. Man and G. Prusty. Neural network modelling for damage behaviour of composites using full-field strain measurements. *Composites Structures*, Vol. 93:pp. 383–391, 2011.

[86] D. W. Marquardt. An algorithm for least-squares estimation of nmiscar parameters. *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11:pp. 431–441, 1963.

[87] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, Vol. 5:pp. 115–133, 2011.

[88] L. McDonald. Florence Nightingale, passionate statistician. *Journal of Holistic Nursing*, Vol. 16:pp. 267–277, 1998.

[89] Y. E. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2$. *Soviet Math*, Vol. 27:pp. 372–376, 1983.

[90] A. Ng. C229 lecture notes: Classification with machine learning. *Department of Data Science and Machine Learning, Stanford*, 2018.

[91] A. Ng. Cs294a lecture notes: Sparse autoencoder. *Department of Data Science and Machine Learning, Stanford*, 2018.

[92] L. T. K. Nguyen and M. Kip. A data-driven approach to nonlinear elasticity. *Computers & Structures*, Vol. 194:pp. 97–115, 2018.

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

[94] A. Oishi and G. Yagawa. Computational mechanics enhanced by deep learning. *Computer methods in applied mechanics and engineering*, Vol. 327:pp. 327–351, 2017.

[95] J. Paik. *Ultimate Limit State Analysis and Design of Plated Structures, Second Edition*. John Wiley Sons Ltd., 2018.

[96] P. Papalambros and D. Wilde. *Principles of optimal design: modeling and computation*. Cambridge University Press, 2000.

[97] B. Pawar. Here are 9 interesting facts about howrah bridge you should read about., 2019. URL `https://www.india.com/travel/articles/9-interesting-facts-about-howrah-bridge-you-should-certainly-read-3240018/`.

[98] M. P. Perrone. Pulling it all together: Methods for combining neural networks. *Advances in neural information processing systems*, pages pp. 1188–1189, 1994.

[99] P. Ramachandran, B. Zoph, and Q. Le. Swish: A self-gated activation function. *Google Brain Research Department*, pages pp. 1–12, 2017.

[100] P. Ramhualli, L. Udpa, and S. S. Udpa. Finite element neural networks for solving differential equations. *IEEE Transactions on Neural Networks*, Vol. 16:pp. 1381–1392, 2005.

[101] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, Vol. 65(6):pp. 386–408, 1958.

[102] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, Vol. 323:pp. 533–536, 1986.

[103] R. E. Schapire. The strength of weak learnability. *Machine Learning*, Vol. 5:pp. 197–227, 1990.

[104] H. Schwenk and Y. Bengio. Boosting neural networks. *Neural Computation, MIT Press*, Vol. 12:pp. 1869–1887, 2000.

[105] G. Seni and J. Elder. *Ensemble Methods in Data Mining: Improving accuracy through combination predictors.* Morgan and Claypool Publishers, 2010.

[106] 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 and Processing Technology*, Vol. 209:pp. 1920–1935, 2009.

[107] S. Shroff. Lecture notes - linear modelling (incl. fem). *TU Delft*, 2017.

[108] J. Sietsma and R. Dow. Neural net pruning - why and how. *International Conference on Neural Networks (IEEE)*, 1988.

[109] J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, Vol. 4:pp. 67–79, 1991.

[110] T. W. Simpson, D. K. J. Lin, and W. Chen. Sampling strategies for computer experiments: Design and analysis. *International Journal of Reliability and Applications*, pages pp. 1188–1189, 2001.

[111] E. Siris. What is osteoporosis and what causes it?, 2019. URL `https://www.nof.org/patients/what-is-osteoporosis/`.

[112] N. Srivastava. Improving neural networks with dropout. *Master Thesis, Department of Computer Science, University of Toronto*, 2013.

[113] N. Srivastava, A. Krizhevsky G. E. Hinton, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, Vol. 15:pp. 1929–1958, 2014.

[114] Z. Su, H. Ling, L. Zhou, K. Lau, and L. Ye. Efficiency of genetic algorithms and artificial neural networks for evaluating delamination in composite structures using fibre bragg grating sensors. *Smart Material Structures*, Vol. 14:pp. 1541–1553, 2005.

[115] I. Sutskever, J. Marthens, G. Gahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28, 2013.

[116] A. Szekrényes and J. Uj. Finite element modelling of the damage and failure in fiber reinforced composites (overview). *Mechanical Engineering*, Vol. 46:pp. 139–158, 2002.

[117] Y. Teboub and P. Hajela. A neural network based damage analysis of smart composite beams. *4th Symposium on multidisciplinary analysis and optimization*, 1992.

[118] C. Thornton, F. Hutter, H. H. Hoos, and L. Leyton-Brown. Auto-weka: Combined selection of hyperparameter optimization of classification algorithms. *Department of Computer Science, University of British Columbia*, 2013.

[119] A. M. Turing. Computing machinery and intelligence. *MIND: A quarterly review of psychology and philosophy*, Vol. 236:pp. 433–460, 1950.

[120] M. T. Valoor and K. Chandrashekhara. A thin composite beam model for delamination prediction by the use of neural networks. *Computer Science and Technology*, Vol. 60:pp. 1773–1779, 2000.

[121] A. Varotsis. Introduction to metal 3d printing., 2018. URL `https://www.3dhubs.com/knowledge-base/introduction-metal-3d-printing/`.

[122] S. Wager, S. Wang, and P. Liang. Dropout training as adaptive regularization. *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, Vol. 1:pp. 351–359, 2013.

[123] S. I. Wang and C. D. Manning. Fast dropout training. *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28, 2013.

[124] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioural sciences. *PhD Thesis, Harvard University*, 1974.

[125] P. Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting (Vol. 1).* John Wiley and Sons, 1994.

[126] WHO. International year of the nurse and the midwife, 2020. URL `https://www.who.int/news-room/campaigns/year-of-the-nurse-and-the-midwife-2020`.

[127] K. Willems. Keras tutorial: Deep learning in python, 2019. URL `https://www.datacamp.com/community/tutorials/deep-learning-python`.

[128] Q. Xu, Y. Yang, Y. Liu, and X. Wang. An improved latin hypercube sampling method to enhance numerical stability considering the correlation of input variables. *IEEE Access*, 5:15197–15205, 2017.

[129] G. J. Yun, J. Ghaboussi, and A. S. Elnashai. Self-learning simulation method for inverse nmiscar modeling of cyclic behavior of connections. *Computational Methods Applied to Mechanical Engineering*, Vol. 197:pp. 2836–2857, 2008.

[130] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2017.

[131] Z. Zhou. *Ensemble Methods: Foundations and Algorithms.* Chapman Hall, 2012.

[132] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals. Sixth Edition.* Elsevier, 2005.