

Delft University of Technology
Materials Science and Engineering
MSc Thesis

**MULTISCALE MODELLING OF LATTICE MATERIALS:
A NOVEL APPROACH USING BEAM NEURAL
NETWORKS**

Author
P.J. van IJzendoorn

[This page is intentionally left blank.]

Delft University of Technology
Materials Science and Engineering
MSc Thesis

**MULTISCALE MODELLING OF LATTICE MATERIALS:
A NOVEL APPROACH USING BEAM NEURAL NETWORKS**

Author

P.J. van IJzendoorn

to obtain the degree of Master of Science
at the Delft University of Technology,
Faculty of Mechanical Engineering.

Student number:	4719921		
Project duration:	Sep, 2023 - Nov, 2024		
Thesis committee:	Dr. I.B.C.M. Rocha	Supervisor	TU Delft, CEG
	M.A. Maia	Daily supervisor	TU Delft, CEG
	T. Gärtner	Daily supervisor	TU Delft, CEG
	Dr. S. Kumar	Supervisor	TU Delft, ME (Chair)
	Dr. F.P. van der Meer		TU Delft, CEG



[This page is intentionally left blank.]

Abstract

A novel surrogate model to approximate microscopic behaviour and accelerate concurrent multiscale finite element simulations is proposed. The study serves as a proof of concept, focusing exclusively on 2D, geometric non-linear lattice materials. Despite numerous successful implementations of surrogate modelling techniques in literature, challenges remain, mainly with the black-box nature of most of these models, suffering from lack of interpretability. To tackle these issues, this study reintroduces physics into the model through the use of beam theory in so-called *Beam Neural Networks*. These networks are tested against a benchmark feed-forward neural network in both interpolation and extrapolation. Although the findings do not satisfy the requirements for practical application, they do indicate that the introduction of beam theory to the model has improved the model's extrapolation ability, suggesting that the proposal has improved robustness and interpretability of the model. Given further optimization, there is promise of Beam Neural Networks to become an useful tool to accelerate concurrent multiscale modelling in the future.

[This page is intentionally left blank.]

ACKNOWLEDGEMENTS

This thesis marks the end of my life as a TU Delft student, starting as a Mechanical Engineering student during my bachelor's degree and ending as a master's student Materials Science and Engineering. A journey that has provided me with the tools to fulfill my ambitions and has encouraged me to develop a more academic mindset. Qualities that will stay with me for a lifetime.

There are many people I need to thank for my time here, but most important of all are my parents, who have always been there for me, during the good and the bad days, who have always believed in me and who have made me who I am today. From the bottom of my heart, thank you. Secondly, I would like to thank my friends: the Brasboys and the wizards of Gee Whiz, you have made my time here unforgettable and have become friends for life. Lastly, I would like to express my gratitude to my thesis supervisors: Iuri, for the opportunity work on this thesis topic and for lending his expertise, your knowledge on the subject feels boundless. Marina and Til, for their unwavering support, their generous amount of time, their patience and their expertise, I truly could not have done this without you. My time in Delft has been exceptional and I am happy to have shared it with all of you.

It has been a privilege to be a part of TU Delft.

*Paul van IJzendoorn
Delft, November 2024*

[This page is intentionally left blank.]

TABLE OF CONTENTS

1	Introduction	1
2	Theoretical background	3
2.1	Lattice materials	3
2.2	Beam theory	5
2.3	Numerical modelling	6
2.3.1	Finite element method	7
2.3.2	FE ² method	12
2.4	Artificial Neural Networks (ANN)	12
2.4.1	Linear regression	13
2.4.2	Feed-Forward Neural Networks (FNN)	14
2.4.3	Recurrent Neural Networks (RNN)	17
2.4.4	The black-box problem	19
2.4.5	Physically Recurrent Neural Networks (PRNN)	20
3	Model architectures	22
3.1	Preliminaries	22
3.2	Feed-Forward Neural Network (FNN)	22
3.3	Beam Neural Networks (BNN)	23
3.3.1	Linear Beam Neural Network (BNN-L)	23
3.3.2	Non-Linear Beam Neural Network (BNN-NL)	24
3.3.3	Finite-Element Beam Neural Network (BNN-FE)	25
4	Data generation	27
5	Model selection and evaluation	29
5.1	Dataset sizes	29
5.2	Hyperparameter characterisation	29
5.3	Learning curves and model selection	30
6	Results and discussion	32
6.1	Hyperparameters	32
6.2	Network learning	35
6.3	Extrapolation	37
6.4	Exploratory studies	43
7	Conclusions and further research	47
	Bibliography	49
A	Appendix: Fixed versus variable beam angles in beam-layer	53
B	Appendix: Sparse versus dense datasets	56
C	Appendix: Select, complete learning curves	58
D	Appendix: Extrapolation for a re-entrant lattice	60
E	Appendix: Extrapolation performance in loading space	65
F	Appendix: BNN-L beam-layer deformations	70
G	Appendix: BNN-NL beam-layer deformations	73
H	Appendix: Exploratory study on BNN-FE with increased number of elements	76
I	Appendix: Exploratory study on BNN-FE with non-linear encoder	78
J	Appendix: Exploratory network performance in loading space	80

[This page is intentionally left blank.]

Throughout history, there has always been a continuous desire for new materials to deal with the challenges introduced by new technologies and needs. Solutions to some of these challenges have been found in so-called lattice materials. These materials consist of a periodic arrangement of beams, occupying a blurry line between structures and materials. Because of this, lattice materials cover an unique range of applications, most notably as metamaterials, which exhibit properties beyond their constituents. Recently, lattice materials have gained more attention in research, not only because of their properties, but also because of advancements in computational science and micro-fabrication. One technique, the FE^2 method, is particularly popular for modelling lattice materials, as it is a multi-scale method that is able to couple the macroscale behaviour of the material to microscale phenomena. Yet, despite advancements in computational science, these methods remain unpractical in full-scale simulations. In an effort to overcome this limitation, machine learning (ML) techniques have been developed to act as a surrogate model replacing the microscale simulations. Although proven effective, shortcomings remain, most notably in their lack of interpretability. To address this, numerous physics-based ML models have been suggested in literature.

In this study, a novel approach of embedding physics into a ML model is presented, based on incorporating beam theory into an Artificial Neural Network (ANN), henceforth referred to as Beam Neural Network (BNN). The main challenge in this work is to develop a network with the specific purpose of accelerating concurrent finite element simulations of geometric non-linear lattice materials, as such material non-linearity is not taken into account. Physics is introduced through in two different approaches: the first is through Euler-Bernoulli beam theory, relating end-point displacements to end-point forces. The second incorporates one-dimensional, finite element models of cantilever beams, solved in static, geometrically non-linear context. The main benefit of the finite element approach is its versatility, establishing a foundation for more sophisticated future applications: most notably, dynamics, as the state of the model is known for each time step: a requirement for incorporating dynamics. Since the primary goal of this work is to serve as a proof of concept, speed comparisons to a benchmark FE^2 simulation are omitted, and complexity is reduced by focusing exclusively on 2D lattice materials, namely honeycomb and re-entrant lattices. The BNNs are instead compared to a Feed-Forward Neural Network (FNN), despite the availability of alternatives. This is because FNNs are universal approximators and well-established in literature.

The data used in this study is obtained from finite element simulations of a single unit-cell with periodic boundary conditions, covering the entire loading space. To assess the model performances, first a hyperparameter selection study is performed, defining the configurable parts of the networks. Next, the models are assessed in interpolation, showcasing their learning efficiency. However, to truly assess the effects of the incorporated physics, network performance in extrapolation is investigated, where the benchmark FNN is expected to degrade rapidly, whereas the physics-based models are expected to demonstrate more effective generalisation. Lastly, in order to justify any shortcomings of the BNNs, their interpretability can be utilised, comparing beam-deformations to the lattice defor-

mations, something that is not possible for the FNN.

In summary, the research objective can be described as such:

The goal of this study is to combine beam theory and ANNs to design a physics-based ML-model (BNN), that offers improved interpretability and extrapolation capability in comparison to a benchmark FNN. The model is designed within the framework of FE^2 multiscale modelling, replacing microscale simulations to enhance computational speed, although this is not investigated: this study is to be regarded as a proof of concept. Expectations are that the proposed BNNs will enhance robustness and interpretability, although the computational costs might restrict practical application to larger FE-models.

The report is structured as follows. In Chapter 2, an overview of important theoretical concepts relevant to the study is provided. The model architectures of the benchmark FNN and the novel BNNs are provided in Chapter 3, while Chapter 4 describes the method used to generate data and specifications on the investigated lattices. Chapter 5 outlines the strategies used to evaluate different networks and to establish which one is the most effective. Results and the discussion are covered in Chapter 6. Conclusions and remarks on future research are provided in Chapter 7.

2 | THEORETICAL BACKGROUND

In this chapter, important theoretical concepts relevant to the study are introduced. Section 2.1 discusses lattice materials, Section 2.2 contains background information on Euler-Bernoulli beam theory, fundamentals on numerical modelling can be found in Section 2.3, while artificial neural networks are introduced in Section 2.4.

2.1 LATTICE MATERIALS

For the context of this work, lattice materials are materials that consist of a periodic arrangement of beams that form unit cells, in either two or three dimensions. The beam cross-sections are considerably smaller than their outer dimensions and can be made from a wide range of materials such as metals, composites, hydrogels and ceramics [1]. Lattice materials are of interest because of their very high structural efficiency, as each beam or rod is loaded in its optimal orientation, this results in a stiff or strong but light material [2, 3]. Applications of these lattice materials have become more mainstream in recent years, not only because of advancements in computational science and micro-fabrication methods but also because of their use as metamaterials. Metamaterials exhibit properties beyond their individual building blocks, due to the way the geometry of the unit cell is constructed, resulting in unique properties [4].

Lattice materials have a unique range of applications because the design of the geometry influences their properties. Examples of such applications are: thermal protection systems due to low thermal conductivity, aerospace applications due to ultra-high stiffness and for use in ceramic satellite parts because of negative thermal expansion behaviour [3]. In this section, the most prominent properties and applications are given.

Ultra-lightweight structures

Composite lattice materials have shown to be able to fill in remarkable gaps in Ashby's chart for material selection, namely that of very stiff and strong, low-density materials [5]. This is shown in Figure 2.1 for the relation between compressive strength and density [6]. Lattice materials are able to achieve this over traditional materials because of their high material efficiency, geometries are chosen in such a way that loads can be distributed efficiently between struts. This strategic placement results in a porous material, having a very high stiffness-to-weight and strength-to-weight ratio [2, 3]. Such ultra-lightweight materials are particularly useful for applications where performance is dominated by the mass of the part, such as aerospace applications. More specifically, the use of lattice materials in sandwich structures (panels) is widespread, generally restricted to 2D-lattice structures, which are mostly deformed in shear and bending when loaded out-of-plane [2]. However, more recently, the use of 3D-lattice cores has been given more attention in research, such as that of A. Alshaer and D. Harland [7]. In this work the authors compared 2D sandwich panels to 3D variants and showed that the 3D-lattice cores are

more stretch-dominated and have excellent normalised stiffness and strength properties, meaning that structures can be more lightweight than 2D counterparts.

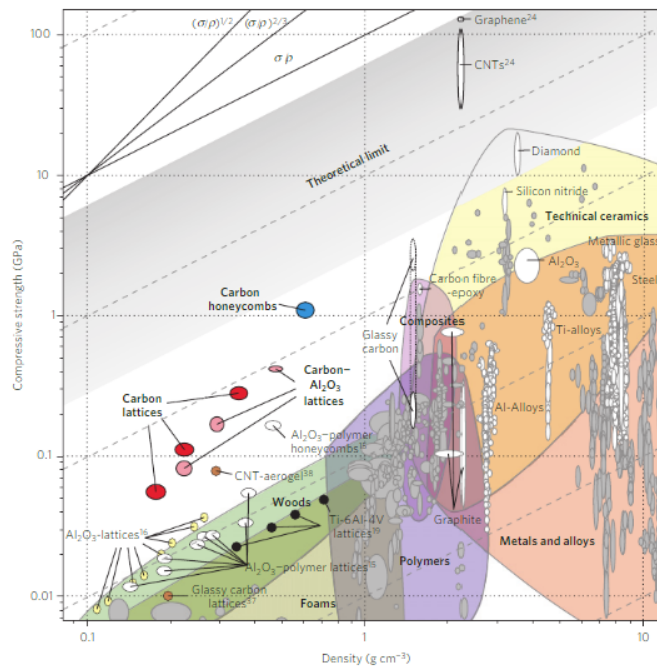


Figure 2.1: Compressive strength-density Ashby chart [6].

Auxetic materials

Auxetic materials are metamaterials that have a negative Poisson's ratio. These materials contract laterally when under compression and expand laterally when under tension [8]. This unusual property might seem contradictory to what is customary in materials science, however auxetic materials do exist through man-made architectures and geometries. Examples of such (2D) structures are given in Figure 2.2 and consist of re-entrant types (Figure 2.2a), which rely on movement of the ribs, chiral types (Figure 2.2b), which are ligaments connected to rings that wind and unwind under movement, similarly, in the missing rib model (Figure 2.2c), the circular core of the chiral type is replaced with a rigid cross [9], rotation squares, relying on the rotation of rigid polygons connected by hinges (Figure 2.2d) and perforated sheets (Figure 2.2e), which may mimic the properties of the rotation squares structure using sheet material [8]. In practice, however, perforated sheets also deform out-of-plane [10].

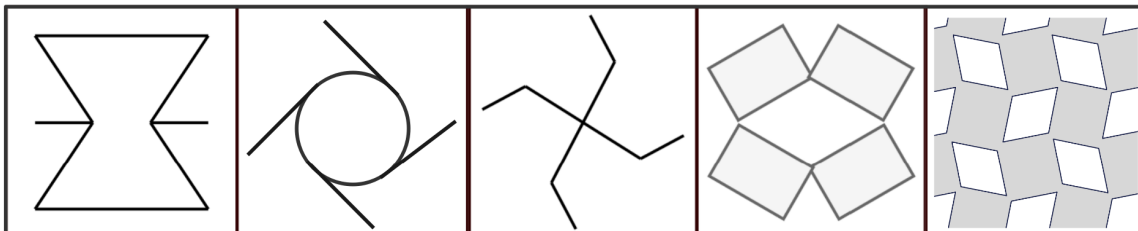


Figure 2.2: Selected types of auxetic material structures, from left to right: a) Re-entrant type, b) Chiral circular, c) Missing rib model, d) Rotation squares, e) Perforated sheet.

Research in auxetic materials has gained traction over recent years, because of their use in many different industries, such as medical, sports and defense. These materials have

high indentation resistance: while most materials move away from the point of contact, auxetic materials move towards the indentation, increasing the apparent density of the material near this point, a property that is very useful for gloves and padding (sports) or protective armour (defense) [3, 8]. A visual of auxetic behaviour is shown in Figure 2.3. Moreover, auxetic materials have increased fracture toughness and shear properties, while also having very remarkable wave propagation properties. Auxetic foams have shown to have 10 times higher damping capacity than conventional foams. Furthermore, auxetic materials have shown to have wave-steering properties, useful for guiding waves [8].

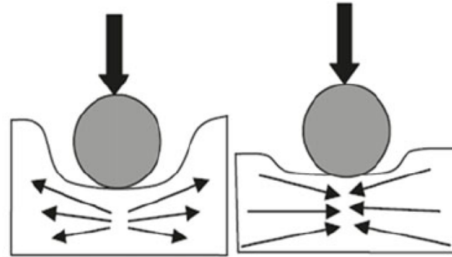


Figure 2.3: Non-auxetic (left) versus auxetic (right) materials' behaviour when indented [11].

Tunable properties

While lattice materials can be used to make very efficient use of the properties of their constituents, they can also be tuned to exhibit desirable properties. A version of this has been explored by X. Wang et al. [12], where plate lattice metamaterials (PLM) were investigated. The authors found that the properties of the PLM were easily tunable when geometric properties such as plate thickness and hole diameter were varied. Changes in elastic moduli, Poisson ratio's and specific energy absorption were observed and linked to the changes in geometric properties. When a systematic approach is used to achieve desirable properties, possibilities open up to use lattice materials in designs that require very specific properties. In this, machine learning methods that aid the design phase can be imagined. These models could assemble different structures of unit cells into a single lattice, in order to achieve a stress-strain relationship that can be custom-tailored to design needs [3].

2.2 BEAM THEORY

This section discusses the beam theory that is used later for implementation in the beam neural network (Chapter 3). More particularly, it discusses Euler-Bernoulli beam theory.

Euler-Bernoulli beam theory, also called *classical beam theory*, was first proposed in the 18th century and, being the first formulated beam theory, set the basis for many advancements in structural engineering. To be able to apply Euler-Bernoulli beam theory, a few assumptions need to be satisfied [13, 14]. The theory is applied to straight beams, without elongation or torsion around the longitudinal axis, deformations are small and in a single plane. Lastly, and perhaps most importantly, Euler-Bernoulli calculations neglect shear deformations, implying that the cross-sectional planes remain perpendicular to the neutral axis of the beam for any deformed state.

The governing equations for Euler-Bernoulli beams in bending in the xz -plane are the following [13, 14, 15, 16]:

$$\phi(x) = \frac{dw(x)}{dx}, \quad (2.1)$$

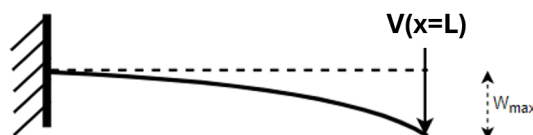
$$M(x) = -EI \frac{d^2w(x)}{dx^2}, \quad (2.2)$$

$$V(x) = -EI \frac{d^3w(x)}{dx^3}, \quad (2.3)$$

$$q(x) = EI \frac{d^4w(x)}{dx^4}, \quad (2.4)$$

where $w(x)$ is the lateral displacement, $\phi(x)$ is the slope of the beam, $M(x)$ is the bending moment in the beam, $V(x)$ is the shear force in the beam, $q(x)$ is a distributed load, E is the Young's modulus and I is the mass moment of inertia. E and I are assumed constant throughout the whole beam.

In order to solve the above equations, boundary conditions must be known. For a cantilever beam, this means that the deflection and the slope of the beam at the fixed end should both be zero, while at the free end, the bending moment is zero and the shear force is equal to the applied load. For common engineering problems, tables of solutions to the Euler-Bernoulli equations are available, these tables can be found in any book on structural engineering. The author employed the book *Mechanics of Materials* by R.C. Hibbeler [17]. It is common practice, however, to describe the deflections of beams in terms of the applied loads. In this study, though, the opposite relation is desired. In Figure 2.4, this relation is depicted for a cantilever beam, describing the maximum deflection.



$$V(x=L) = \frac{3EIw_{max}}{L^3} \quad (2.5)$$

Figure 2.4: Cantilever deflection example for a Euler-Bernoulli beam.

For a beam in tension or compression, the extension or compression can be described by:

$$EA \frac{du_x(x)}{dx} = N_x(x), \quad (2.6)$$

where N_x is a tensile force. Because of infinitesimal displacements, linear superposition of pure bending and pure tension/compression can then be applied to obtain the total displacement field of a beam [13].

2.3 NUMERICAL MODELLING

Predicting the mechanical behaviour of materials using constitutive relations is highly complex, especially when plasticity and other non-linearities are introduced, for example due to strut-buckling in lattice materials [18, 19]. A common way to work around these

complexities is through numerical simulations, as these are able to reliably and accurately predict the mechanical properties of the material as a whole. It must be noted that, where Euler-Bernoulli beam theory does not take into account deformation shear effects, the numerical methods in this work do take these into account by employing the Timoshenko beam model, i.e. instead of using shear-stiff Euler-Bernoulli beams, shear-flexible Timoshenko beams are used [20]. This section outlines the most important numerical methods used for modelling materials, starting with the finite element method in Section 2.3.1 and thereafter its multiscale extension, the FE² method in Section 2.3.2.

2.3.1 Finite element method

The finite element method (FEM) is a numerical analysis technique that is used to approximate a solution by dividing the domain into subdivisions of finite elements. It is a computationally expensive method, as it involves approximating a large number of partial differential equations and solving the problem (numerically) in weak form. Therefore, this method has found more mainstream usage as the computational power of computers has grown [21]. There are many different methods and interpretations of finite element methods. This section will discuss only the most relevant and mainstream methods to solve static finite element analyses, for linear and non-linear systems.

In this work, only a global review of the theory and equations behind the methods will be given, for a more in depth approach to these methods, the reader is referred to the books of K.J. Bathe (1996) [22] and R.D. Cook (1989) [23], as the theory in this section is based upon these sources. Both books still, to this day, give a great overview of the possibilities of FEM and the theoretical framework behind it. Besides these two books, a more recent review of methods by B. Yang (2019) [24] has been used.

2.3.1.1 Linear analysis

The most common type of analysis is a linear, static system. For these analyses, the response of the system is proportional to the applied loads, meaning that the stiffness matrix, which describes the relation between stresses and strain in the material, remains constant for the duration of the analyses. In this situation, deformations and rotations are normally small and material properties are constant. The governing equation describing this situation is:

$$\mathbf{f} = \mathbf{K}\mathbf{u}, \quad (2.7)$$

where \mathbf{f} is the load vector, \mathbf{K} is the global stiffness matrix and \mathbf{u} is the displacement vector. The global stiffness matrix is obtained through assembly of the local, element stiffness matrices. The solution of the equilibrium equation (Equation 2.7), is then obtained, which will return an approximation of the response of the full structure under loading. In case of structural analyses these will be the nodal displacements and stresses. The assembly of Equation 2.7 is done in the following way:

First, the entire domain of the problem will be discretised into smaller parts, called *finite elements*. Each element is defined by *nodes* along the element's boundaries that serve as a connection to other elements, nodes are interconnected to other nodes through *edges*, to define the element shapes and create *surfaces*. The shape of these elements may vary per application, a list of element types and applications is given in the work of A.F. Bower (2018) [25]. Interpolation functions, also known as shape functions, are then introduced

to calculate an approximation of the field variable (displacements), $\tilde{\mathbf{u}}(x, y, z)$, within each element:

$$\tilde{\mathbf{u}}(x, y, z) \approx \sum_{i=1}^{n_{nodes}} \mathbf{N}_i \mathbf{u}_i(x, y, z), \quad (2.8)$$

where $\mathbf{u}(x, y, z)$ are the nodal displacements and \mathbf{N} are the interpolation functions. The choice of interpolation function and its polynomial order depend on the element geometry, boundary conditions and the number of nodes within the element.

From the nodal displacements, element strains and stresses can be approximated, assuming linear-elastic material behaviour:

$$\boldsymbol{\epsilon} = \mathbf{B} \mathbf{u} \quad (2.9)$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\epsilon}, \quad (2.10)$$

where \mathbf{B} is the strain-displacement matrix. In this work, \mathbf{B} is defined specifically for beams:

$$\mathbf{B} = \begin{bmatrix} \cos(\phi)N_1' & \sin(\phi)N_1' & 0 & \cos(\phi)N_2' & \sin(\phi)N_2' & 0 \\ -\sin(\phi)N_1' & \cos(\phi)N_1' & -N_1 & -\sin(\phi)N_2' & \cos(\phi)N_2' & -N_2 \\ 0 & 0 & N_1' & 0 & 0 & N_2' \end{bmatrix}, \quad (2.11)$$

where ϕ is the angle between the element and the global x-axis and N_1 and N_2 are shape functions. \mathbf{D} is the elasticity matrix, representing the stress-strain relation for linear elasticity, for beams only:

$$\mathbf{D} = \begin{bmatrix} EA & 0 & 0 \\ 0 & GA_s & 0 \\ 0 & 0 & EI \end{bmatrix}, \quad (2.12)$$

where E and G are the Young's modulus and shear modulus of the material respectively. I is the second moment of inertia, A the cross-sectional area and A_s the cross-sectional area corrected with the cross-section dependent shear factor. Applying the virtual work principle, in which the strain energy must equal the work done by nodal forces, you end up back at the governing equation, but now applied separately for each element:

$$\mathbf{K}^e \mathbf{u}^e = \mathbf{f}^e, \quad (2.13)$$

$$\mathbf{K}^e = \int_{\Omega} \mathbf{B}^T \mathbf{D} \mathbf{B} d\Omega. \quad (2.14)$$

The elemental stiffness matrix, \mathbf{K}^e , is calculated using Equation 2.14. In this equation, Ω represents the domain of the element, consisting of integration points at which the calculations are performed. The elemental stiffness matrices are then mapped from the local system to the global system and assigned to the global stiffness matrix. The force vector consists of body forces, surface tractions, concentrated nodal forces and the effects of initial stresses and strains. Finally, the equilibrium equations of the whole system, Equation 2.7, are obtained.

In order to analyse the stress and strain distribution in the whole system, the equilibrium equations are to be solved. For a static, linear system, there are two main methods to do this, either a direct solution method based on Gauss elimination or an iterative solution method. For Gauss elimination, the stiffness matrix is formed into the upper-triangular form and solved through back-substitution. Iterative methods converge towards the solution until a certain tolerance is met. For more specifics on these methods the reader is redirected to the work of K.J. Bathe (1996), Chapter 8 [22].

2.3.1.2 Non-linear analysis

While linear, static problems have been discussed in Section 2.3.1.1, finite element methods are also very popular when non-linearities are present, as analytical solutions for these problems can quickly become very complex. In structural analyses, non-linearities result in displacement-dependent stiffness matrices and load vectors:

$$\mathbf{f}(\mathbf{u}) = \mathbf{K}(\mathbf{u})\mathbf{u}. \quad (2.15)$$

The cause of non-linearity can be both due to material non-linearity and geometric non-linearity. Material non-linearity expresses itself in plasticity of the material, while geometric non-linearity is due to large deflections or rotations in the system. Even though linear analyses are sufficient for many applications, non-linear analyses have to be performed for situations in which these effects can no longer be ignored in order to get meaningful results. In this section, the differences between linear and non-linear static analyses will be discussed.

This study will not take into account the effects of material non-linearity, instead it will only focus on the effects of geometric non-linearity, therefore, in the case of a one-dimensional beam, the stiffness matrix can be constructed as:

$$\mathbf{K}^e = \mathbf{K}_M^e + \mathbf{K}_G^e, \quad (2.16)$$

where \mathbf{K}_M^e contains the linear material relations, from Equation 2.14, although the strain-displacement matrix \mathbf{B} is redefined, for linear elements, as:

$$\mathbf{B} = \begin{bmatrix} \cos(\omega)N_1' & \sin(\omega)N_1' & N_1\gamma & \cos(\omega)N_2' & \sin(\omega)N_2' & N_2\gamma \\ -\sin(\omega)N_1' & \cos(\omega)N_1' & -N_1(1+\epsilon) & -\sin(\omega)N_2' & \cos(\omega)N_2' & -N_2(1+\epsilon) \\ 0 & 0 & N_1' & 0 & 0 & N_2' \end{bmatrix}, \quad (2.17)$$

where ω is a measure of the current orientation of the axis, taking into account the angle between the element and the global x-axis, ϕ , the rotation of the beam axis, ψ , and the shear strain γ , which is a measure of the rotation of the plane normal to the beam axis with respect to the rotation of the beam axis itself. This can be summarised as:

$$\omega = \phi + \psi + \gamma \quad (2.18)$$

or visualised as in Figure 2.5.

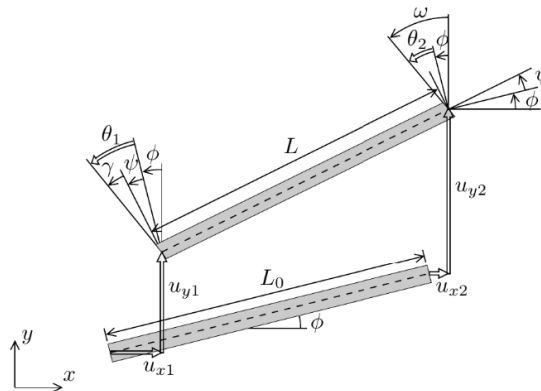


Figure 2.5: Geometrically non-linear element [26].

\mathbf{K}_G^e , is calculated as:

$$\mathbf{K}_G = \int_{L_0} N\mathbf{W}_N + V\mathbf{W}_V dx, \quad (2.19)$$

where N and V are the normal force and the shear force contribution and \mathbf{W}_N and \mathbf{W}_V are defined as [26]:

$$\mathbf{W}_N = \frac{1}{L_0} \begin{bmatrix} 0 & 0 & N_1 \sin(\omega) & 0 & 0 & N_2 \sin(\omega) \\ 0 & 0 & -N_1 \cos(\omega) & 0 & 0 & -N_2 \cos(\omega) \\ N_1 \sin(\omega) & -N_1 \cos(\omega) & -N_1^2 L_0 (1 + \epsilon) & -N_1 \sin(\omega) & N_1 \cos(\omega) & -N_1 N_2 L_0 (1 + \epsilon) \\ 0 & 0 & -N_1 \sin(\omega) & 0 & 0 & -N_2 \sin(\omega) \\ 0 & 0 & N_1 \cos(\omega) & 0 & 0 & N_2 \cos(\omega) \\ N_2 \sin(\omega) & -N_2 \cos(\omega) & -N_1 N_2 L_0 (1 + \epsilon) & -N_2 \sin(\omega) & N_2 \cos(\omega) & -N_2^2 L_0 (1 + \epsilon) \end{bmatrix}, \quad (2.20)$$

$$\mathbf{W}_V = \frac{1}{L_0} \begin{bmatrix} 0 & 0 & N_1 \cos(\omega) & 0 & 0 & N_2 \cos(\omega) \\ 0 & 0 & N_1 \sin(\omega) & 0 & 0 & N_2 \sin(\omega) \\ N_1 \cos(\omega) & N_1 \sin(\omega) & -N_1^2 L_0 \gamma & -N_1 \cos(\omega) & -N_1 \sin(\omega) & -N_1 N_2 L_0 \gamma \\ 0 & 0 & -N_1 \cos(\omega) & 0 & 0 & -N_2 \cos(\omega) \\ 0 & 0 & -N_1 \sin(\omega) & 0 & 0 & -N_2 \sin(\omega) \\ N_2 \cos(\omega) & N_2 \sin(\omega) & -N_1 N_2 L_0 \gamma & -N_2 \cos(\omega) & -N_2 \sin(\omega) & -N_2^2 L_0 \gamma \end{bmatrix}, \quad (2.21)$$

where ϵ is the axial strain and γ is the shear strain, defined as:

$$\epsilon = \frac{\partial u_x}{\partial x}, \quad (2.22)$$

$$\gamma = \frac{\partial u_y}{\partial x} - \theta, \quad (2.23)$$

where u_x and u_y are axial and lateral deformations and θ is the rotation of the beam.

The proposed method to solve a non-linear system in this work is an incremental-iterative solution procedure employing the Newton-Raphson method.

In *incremental-iterative solution procedures*, specifically for this study *displacement control*, target displacements are applied to constrained nodes. During a sequence of iterations, employing the *Newton-Raphson method*, the applied loads are adjusted such that the target displacements are reached. Repeating the process for an entire load-displacement curve, the equilibrium path can be traced, as long as convergence is obtained. A schematic of the Newton-Raphson method can be seen in Figure 2.6, while the incremental-iterative solution procedure is described in Algorithm 1.

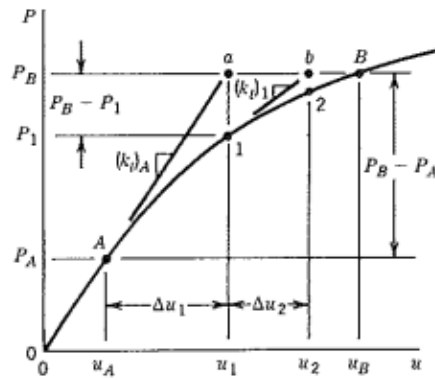


Figure 2.6: The Newton-Raphson method [22].

A few things are important to note in Algorithm 1. Firstly, the initial state, u^0 , should be a reasonable guess of the state, for many applications $u^0 = 0$ is appropriate. Secondly, calculations of the internal force vector and stiffness matrix are performed as described in the equations earlier in this section. Numerical solving of the integrals in Equations 2.14 and 2.19 can be performed effectively using single integration point Gauss integration, which also mitigates *shear locking*: a phenomenon that occurs when linear elements are not able to accurately model bending, causing an additional shear stress to be introduced, making the element appear stiffer than it actually is. Thirdly, for this method a clear distinction between free (f) and constrained (c) DOFs is crucial. In operation 6, the stiffness matrix is constrained in such a way that a new state of the constrained DOFs is forced. Next, during a sequence of iterations, the free DOFs are assigned a new solution (operation 10), however in operation 13 the displacement increment at the constrained DOFs is set to zero, to ensure that the displacements remain as described. Lastly, the residual is calculated only at the free DOFs, as the residual at the prescribed DOFs is inherently zero.

Algorithm 1 Iterative solving algorithm: displacement control [26].

Require: Non-linear relationship $\mathbf{f}_{int}(u)$ and $\mathbf{K}(u)$

- 1: **Initialisation** $n = 0$, $u^0 = 0$, partition DOFs in free (f) and constrained (c)
 - 2: **while** $n <$ number of time steps **do**
 - 3: Get external force vector \mathbf{f}_{ext}^{n+1} { $\mathbf{f}_{ext}^{n+1} = 0$ at free nodes}
 - 4: Initialise new state at previous one: $\mathbf{u}^{n+1} = \mathbf{u}^n$
 - 5: Compute internal force vector $\mathbf{f}_{int}^{n+1}(\mathbf{u}^{n+1})$ and stiffness matrix $\mathbf{K}^{n+1}(\mathbf{u}^{n+1})$
 - 6: Constrain \mathbf{K}^{n+1} such that $\Delta \mathbf{u}_c = \bar{\mathbf{u}}^{n+1} - \bar{\mathbf{u}}^n$
 - 7: Evaluate residual at free DOFs: $\mathbf{r} = \mathbf{f}_{ext,f}^{n+1} - \mathbf{f}_{int,f}^{n+1}$
 - 8: **while** $|\mathbf{r}| >$ Tolerance **do**
 - 9: Solve linear system of equations $\mathbf{K}^{n+1} \Delta \mathbf{u} = \mathbf{r}$
 - 10: Update state: $\mathbf{u}^{n+1} = \mathbf{u}^{n+1} + \Delta \mathbf{u}$
 - 11: Compute internal force vector $\mathbf{f}_{int}^{n+1}(\mathbf{u}^{n+1})$ and stiffness matrix $\mathbf{K}^{n+1}(\mathbf{u}^{n+1})$
 - 12: Evaluate residual at free DOFs: $\mathbf{r} = \mathbf{f}_{ext,f}^{n+1} - \mathbf{f}_{int,f}^{n+1}$
 - 13: Constrain \mathbf{K}^{n+1} so that $\Delta \mathbf{u}_c = 0$
 - 14: **end while**
 - 15: Proceed to next time step: $n = n+1$
 - 16: **end while**
 - 17: **return** Internal force vector \mathbf{f}_{int} and state \mathbf{u}
-

One way to model a lattice material, and perhaps the most straight-forward approach, is to model the entire structure in finite elements, also known as *direct numerical simulation* or *full field analysis*. As much as this is a desirable approach, it is often not feasible due to limitations in computational power [27].

2.3.2 FE² method

The FE² method, as the name implies, is a modelling method in which two FE-models, one at the macroscale, the other at the microscale, are solved in the same computation. This assumes that the principle of scale separation is satisfied, i.e. $L_{micro} \ll L_{macro}$, or in words, that the scale of the micro-model fluctuations must be much smaller than the macro-model fluctuations [28]. While solving the macroscale model, at each integration point a microscale model is solved, this information is then passed back to the macroscale model in an iterative loop. To solve the microscale models, the strain from the macroscale model is downscaled to the micro-model, taking into account periodicity of the RUC, this micro-model is then solved and the homogenized stresses are then upscaled back to the macroscale model. This is repeated for every integration point and at every time-increment of the macro-model [29, 30, 31]. A schematic of the method, applied to fiber-reinforced composites, is shown in Figure 2.7.

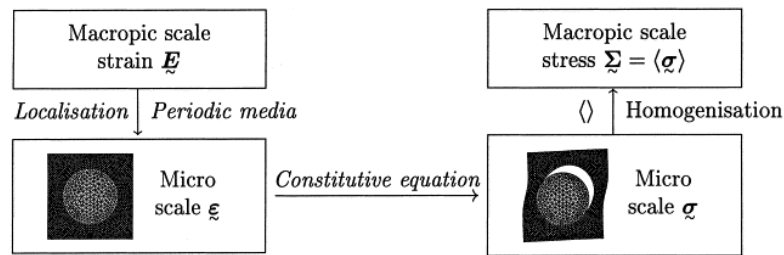


Figure 2.7: Schematic representation of the FE² method [30].

Even though the FE² method removes the need for constitutive laws at the macroscale to describe, for example, non-linear behaviour, because these properties are directly solved from the microscale model, the computational effort of this method is still impractical [31]. Machine learning techniques have been widely adopted in literature to accelerate and reduce the computational cost of these simulations by replacing the microscale model altogether with so-called surrogate models. In this work, focus is put on artificial neural networks (ANNs).

2.4 ARTIFICIAL NEURAL NETWORKS (ANN)

Artificial neural networks are mathematical structures inspired by biological neural networks. Biological neural networks are believed to consist of approximately 10^{10} neurons [32], which are their fundamental building blocks. Artificial neural networks mimic the structure of biological neural networks, where the neurons represent nodes, which are interconnected with edges. Typically, and for the purpose of this study, the nodes are placed in layers, where the first layer is the input layer, the location where values for different features are set, and the last layer is the output layer, where the predictions are obtained. Between the input and output layers, there are hidden layers. These layers can be architected in different ways, but aim to return the predictions as accurate as possible.

Accuracy is obtained by training the network, for which snapshots of the high-fidelity solutions are required. The amount of training data needed depends on the type of problem: static or dynamic, linear or non-linear and the history-dependency of the material. In general, more training data leads to more accurate results. Training can be done *supervised* or *unsupervised* [33]. In supervised learning, the input and the output of the training data are paired, the network is fine-tuned based on the error between the network-generated output and the known output. In unsupervised learning, inputs and outputs are not paired, the network trains itself to uncover trends within the data. This work will focus on supervised learning. A good practice is to divide the data in three different sets [33]: a training set, which is used to fit the learnable parameters of the network, a validation set, which is used to tune the hyperparameters, (e.g. the number of nodes in hidden layers or the learning rate, by comparing the validation loss), and a test-set, which assesses the performance of the network, after training has been completed.

One important thing to remember when designing neural networks, is that a neural network can only be as good as their training data, data should thus be accurate and representative.

The contents of this chapter are mostly based on the books of C.M. Bishop (2006) [33], (2024) [34] and K.P. Murphy (2022) [35]. In this chapter, important theoretical concepts to understand neural networks are discussed. Section 2.4.1 describes the concept of linear regression, the basis of neural network modelling. Feed-forward neural networks are discussed in Section 2.4.2, as they are the most common type of model. Furthermore, recurrent neural networks will be introduced in Section 2.4.3, whereafter the black-box problem is given elaboration in Section 2.4.4. A more complex architecture, the physically recurrent neural network, is introduced in Section 2.4.5.

2.4.1 Linear regression

Linear regression is a supervised machine learning method that is commonly used to predict dependent variables, from now on called targets t , based on a collection of inputs. This section will discuss the theoretical framework behind regression model, to get a better understanding of artificial neural networks (ANNs).

Multivariable linear regression can be expressed as:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + \dots + w_Dx_D. \quad (2.24)$$

In these equations, \mathbf{w} is a collection of weights or regression coefficients and \mathbf{x} are input variables. The weight determines the effect of the input on the output. The larger the weight, the more effect that input has on the output.

For systems with only one input variable, Equation 2.24 reduces to the *simple linear regression* formula $y(x, \mathbf{w}) = w_1x + w_0$, while the multiple input variable method of Equation 2.24 is also known as *multiple linear regression*. When there are multiple outputs, which is called *multivariate linear regression*, Equation 2.24 can be rewritten into:

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}. \quad (2.25)$$

While these methods are used in a plenitude of applications, fitting of the data will only be possible by linear functions. To broaden the usability of the regression method, basis

functions $\phi(x)$ are introduced, which are non-linear functions, for example in polynomial expansion $\phi = [x, x^2 \dots x^j]$. Substituting this in Equation 2.25 gives:

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}). \quad (2.26)$$

Linear regression, therefore, consists of linear combinations of fixed basis functions. However, *the curse of dimensionality* reduces the practical applicability of the method. The curse of dimensionality is a phenomenon in which as the number of features in a system increases, the dimensionality of the problem increases exponentially and the data becomes less densely distributed, making fitting more difficult. Not only would this result in the need for more training data, the final model would also be too complex, fitting to noise rather than the underlying pattern, as the introduced features might not contain useful information. Therefore, not only is accuracy reduced, the computational time is increased.

2.4.2 Feed-Forward Neural Networks (FNN)

The simplest version of a neural network is the feed-forward neural network (FNN), an example of a FNN with a single hidden layer is shown in Figure 2.8. In this model, the weights and biases are adapted throughout the training in the following way:

The first layer is the input layer, every node contains one feature, of in total D features: $x_1 \dots x_D$. Layers 2 to $n - 1$ are the hidden layers, the nodes in these layers form a linear combinations of the previous layer's nodal values, known as the *weighted sum*, z :

$$z_j = \sum_{i=1}^M w_{ji}^{(n)} x_i + w_{j0}^{(n)}, \quad (2.27)$$

where M is the number of nodes in the previous layer, K is the number of nodes in the current layer, $j = 1, \dots, K$, n is the layer number and w_{j0} is the bias, a systematic deviation of the model.

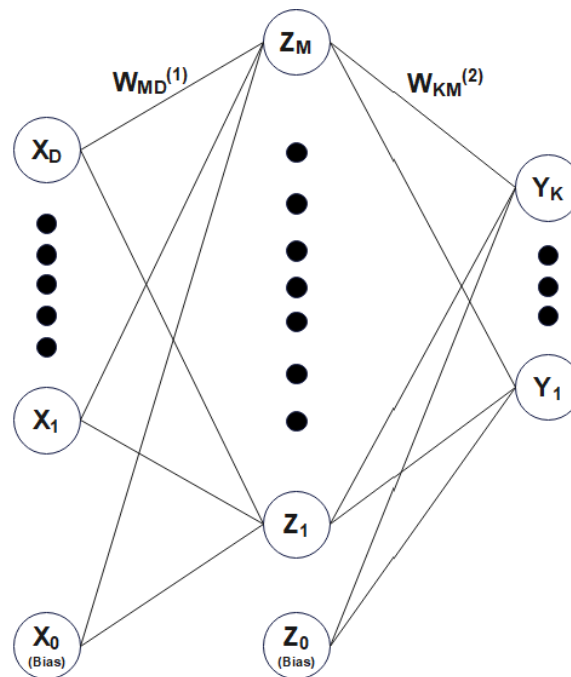


Figure 2.8: Schematic of a neural network containing one hidden layer.

Normally, a non-linear activation function $h(\cdot)$ is then used to introduce non-linearity into the system. The result is the activation a_j , which represents a nodal value in the network:

$$a_j = h(z_j). \quad (2.28)$$

Figure 2.9 shows the process of non-linear regression for one node only.

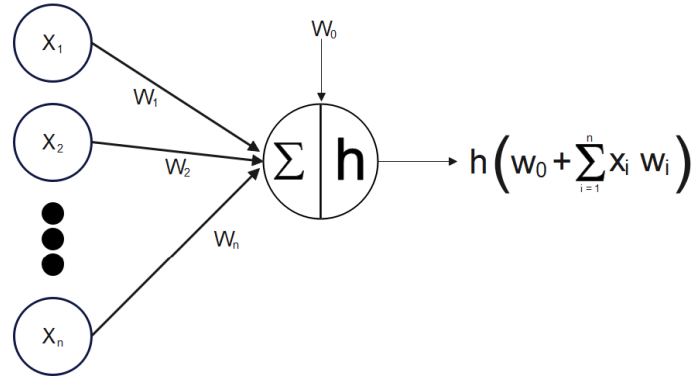


Figure 2.9: Schematic of a non-linear regression process for one node only.

Some common activation functions, the ones used in this study, are shown in Figure 2.10, the ELU, tanh and softplus functions. A large number of variants of these functions exist, most of these activation functions *squash* the input to an output that is bounded between 0 and 1 or between -1 and 1.

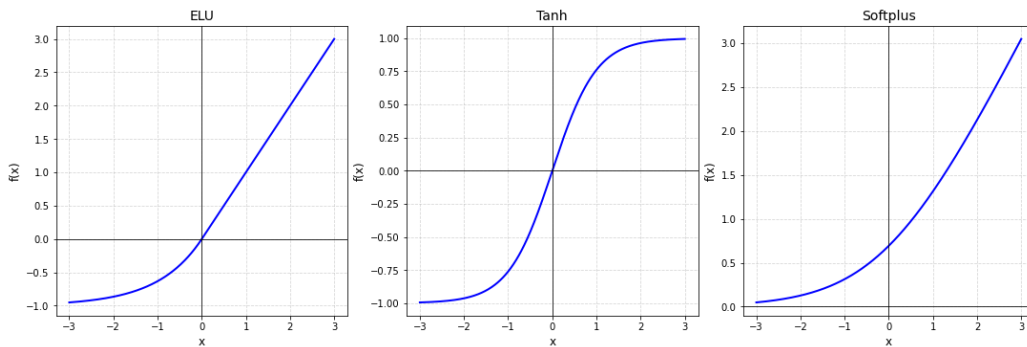


Figure 2.10: ELU, tanh and softplus activation functions.

The process illustrated in Figure 2.9 is repeated for every node in every hidden layer. Noting that the biases are included within the weight vectors, the feed-forward neural network of Figure 2.8, can mathematically be represented as:

$$y_k(\mathbf{x}, \mathbf{w}) = h\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right). \quad (2.29)$$

2.4.2.1 Training

To minimise the error, the network is calibrated such that the weight values are optimized [33]. This is done using a training set, where inputs and outputs (targets t_n) are known. During training of the system, the goal is to minimise the error function, which could be,

for example, the root mean squared error, the mean absolute error or the mean squared error (Equation 2.30):

$$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2. \quad (2.30)$$

One approach to determine the optimal set of weights is to create a multi-dimensional graph of the error function, a surface over weight space. $E(\mathbf{w})$ is a smooth, continuous function, where minima can be found in which the gradient of the error function equals zero:

$$\nabla_{\mathbf{w}}E(\mathbf{w}) = 0. \quad (2.31)$$

Moving in the direction of $-\nabla_{\mathbf{w}}E(\mathbf{w})$, will guide the solution towards the local minimum. The global minimum, the solution with the smallest value for the error function, can be found done by comparing different values of local minima. There is no guarantee, however, that the found minimum is the actual global minimum. For many applications of neural networks, though, finding the true global minimum may not be necessary and comparing several local minima is sufficient. A practical, numerical method to find local minima is gradient descent.

2.4.2.2 Gradient descent

In gradient descent, weights are updated according to [33, 35]:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla_{\mathbf{w}}E(\mathbf{w}^{(\tau)}). \quad (2.32)$$

In this equation, τ represents the iteration step and η is the *learning rate*. Large values for the learning rate might converge faster, but could converge to a globally, sub-optimal solution. Whereas small values for the learning rate require more training epochs, but tend to converge to a more optimal solution. Every iteration step, the weight vector is moved in the direction that reduces the error the most. One thing to consider, however, is that for every step, the entire training set needs to be processed. More efficient methods exist, such as the conjugate gradient and quasi-Newton methods or the use of mini-batches. Mini-batch gradient descent is a method where the training data is divided into mini-batches with a fixed amount of training snapshots. For every epoch the network is trained and updated after each mini-batch is processed. This way, the full amount of training data is still being processed, but the network is updated less frequently. Mini-batch gradient descent allows for more robust convergence and is computationally more efficient.

2.4.2.3 Error Backpropagation

To adjust weight values so that the error is minimised, advantage is taken of the gradient of the error function described in Section 2.4.2.2 using a method called backpropagation [36, 37]. In this method, for each forward pass through the network, a backwards pass is performed to adjust the model parameters. The system therefore receives *feedback* on its performance.

To calculate the gradient of the error function, a network of L layers is considered. An untrained model will take an input, set weights and biases according to an initialisation and, very likely, will return a large error at the output. To reduce this error, it is important to know how the error varies with changes to the weights, more specifically for a single

weight value in layer L . The set of activations in each layer depend on the weights and biases associated with that layer and the activations in the previous layer. This weighted sum is then inserted in an activation function. This process has already been summarised in Equations 2.27 and 2.28. Note that the gradient of the error function depends on the weight, the biases and the activation functions chosen. More specifically, a change in the weights, w , will cause a change in the weighted sum z , which will influence the activation a of which an error can be determined in the output layer.

This process can be summarised in the following equations, starting with the relatively simple interaction between two nodes, one in the output layer L , and one in the previous layer $L - 1$:

$$\frac{\partial E}{\partial w^{(L)}} = \frac{\partial E^{(L)}}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}. \quad (2.33)$$

Based on Equations 2.27 and 2.28, the partial derivatives can be defined as:

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \quad (2.34) \quad \frac{\partial a^{(L)}}{\partial z^{(L)}} = h'(z^{(L)}) \quad (2.35) \quad \frac{\partial E}{\partial a^{(L)}} = \frac{2}{N}(a^{(L)} - y). \quad (2.36)$$

Equation 2.36 is for a network that uses the mean squared error as loss function. A simplified version of Equation 2.33 can then be obtained by replacing the individual partial derivatives with the expressions in Equations 2.34, 2.35 and 2.36:

$$\frac{\partial E}{\partial w^{(L)}} = a^{(L-1)} h'(z^{(L)}) \frac{2}{N} (a^{(L)} - t). \quad (2.37)$$

2.4.2.4 Optimizers

Optimizers exist for the purpose of minimising the error function based on the models learnable parameters. Some revolve around statistics (Bayesian optimization), some are inspired by natural phenomena (Particle Swarm optimization), however most use gradient descent, as explained in Section 2.4.2.2.

Within gradient descent algorithms, the most popular, and the one used in this work, is the Adam optimizer. In the Adam optimizer, momentum and adaptive learning rates are implemented. Momentum entails that, to a certain extent, the direction of previous update is retained, allowing for faster learning. Adaptive learning rates, meanwhile, allows the model to change the learning rates for each parameter based on the history of gradients for that parameter. Both these features allow this optimizer to rapidly converge to an accurate solution. The governing equation for the Adam optimizer is [38]:

$$w_{t+1} = w_t - \eta \frac{v_t}{\sqrt{s_t + \epsilon}} g_t, \quad (2.38)$$

where η is the learning rate, v_t , s_t and g_t are the exponential average of gradients, the exponential average of the squares of the gradients and the gradient at time t along w_j respectively. ϵ is a small value to ensure that division by zero is not possible.

2.4.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are very similar to feed-forward neural networks, however RNNs are used to handle sequential data. Because of this property, RNNs are

used to model materials that show history-dependent behaviour or in dynamics, as the previous state of the network is stored and utilised to predict the current state. This section will discuss the architecture of recurrent neural networks, its possibilities and (dis)advantages.

2.4.3.1 Basic architecture

Basic RNNs employ the same base architecture as FNNs, as discussed in Section 2.4.2, both consist of an input layer, hidden layers and an output layer. The differences however, are firstly that the input vectors of RNNs are sequences of vectors through time [39, 40]: $\mathbf{x} = [\dots, \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, \dots]$, where $\mathbf{x}_t = [x_1, x_2, \dots, x_D]$, and secondly that the values of the nodes within the hidden layers of the previous time step are taken into account when calculating the same values of the current time-step. A *folded* representation of a simplistic RNN is shown in Figure 2.11a, however more insight can be obtained using the *unfolded* representation, shown in Figure 2.11b.

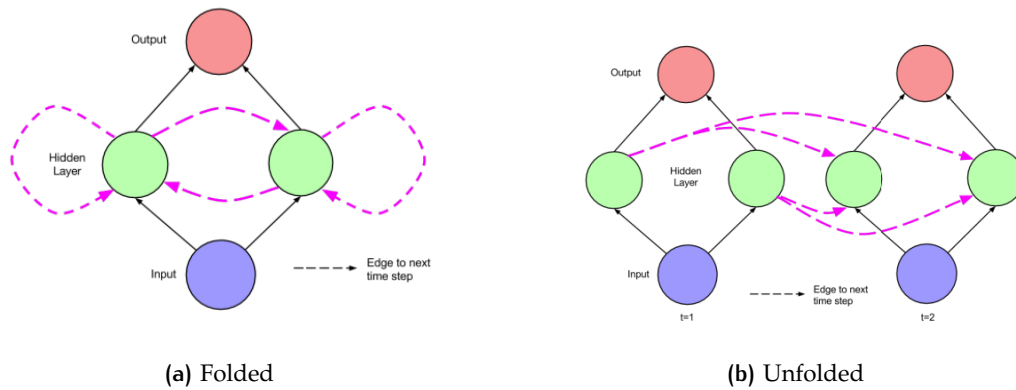


Figure 2.11: Folded and unfolded Recurrent Neural Networks [40].

The unfolded schematic, Figure 2.11b, shows how at every time t , nodes in the hidden layer receive both inputs from the current data \mathbf{x}_t and from hidden node values from the previous state, \mathbf{a}^{t-1} . The inputs from the hidden node values are multiplied by certain, trainable, weights: w_{HH} .

$$\mathbf{a}^t = h(w_{HX}\mathbf{x}^t + (w_{HH}\mathbf{a}^{t-1} + w_0)). \quad (2.39)$$

Equation 2.39 shows how values for hidden nodes are determined, where h is the activation function, and w_0 is the bias [39, 40].

2.4.3.2 Network properties

Just like regular FNNs, training is usually done with gradient descent-based methods that minimise some error-function, however instead of applying backpropagation, a new method *backpropagation through time (BPTT)* is introduced [39, 40, 41]. BPTT methods unfold the RNN, where parameters (weights and biases) of each separate copy in time are shared, then gradients are determined for each time step, capturing the dependencies over time. During backpropagation in RNNs, however, the concept of *vanishing and exploding gradients* may become an issue [39, 40, 41]. The problem of vanishing gradients can be described as the exponential shrinkage of error gradients, due to the continuous propagation back through time, making the network more dependent on short-term contributions than long-term ones. The problem of exploding gradients would be the polar opposite,

where there is a stronger dependence on long-term contributions than short-term ones. This issue will limit the memory capacity of the RNN. More advanced *Long Short Term Memory (LSTM)* architectures are able to deal with this. These networks utilise *memory blocks* and *memory cells*, which regulate the flow of information. Furthermore, RNNs are severely restricted by the curse of dimensionality, as the increased number of dependencies leads to an exponential increase in the amount of training data required. Lastly, RNNs still deal with the issue of overfitting data, as well as FNNs, reducing their generalisation properties. Therefore, while RNNs have great potential to learn sequential dependencies in data, the method still contains pitfalls which may affect data-interpretability. As such, RNNs are considered black-box models, as no physical insight is utilised during data analysis.

2.4.4 The black-box problem

Feed-forward neural networks and recurrent neural networks have been used extensively in predicting the stress-strain relations of materials. As FNNs are unable to keep track of the loading-history of the material, they can only, realistically, be used to analyse elastic material behaviour [42, 43]. History-dependent materials could, theoretically, be analysed, however it would require the introduction of extra history variables, which grow the feature space and amount of training data needed. To handle history-dependence more efficiently, recurrent neural networks were introduced, which are able to model sequential data and therefore have some sense of history. These models are capable of modelling plasticity, as can be seen in Figure 2.12a, however these models can only approximate accurately for problems within the range of training data, Figure 2.12b: extrapolation to untrained loading conditions is not accurate [42, 44]. Therefore, a lot of data, with different loading conditions and histories, is required to cover the loading-space.

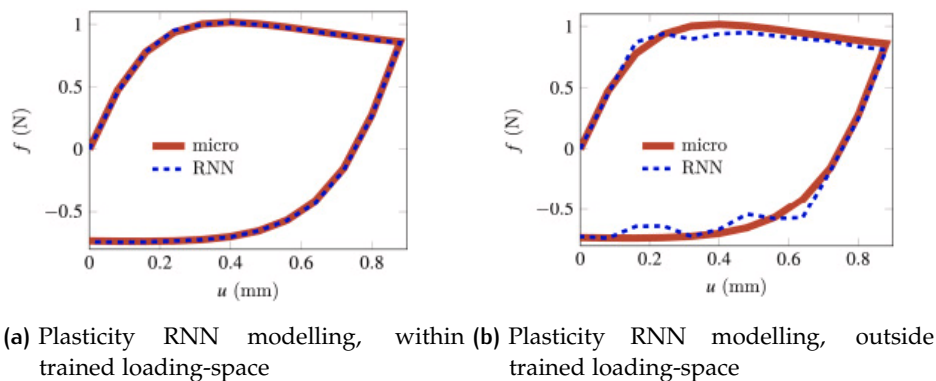


Figure 2.12: Modelling plasticity with RNN [44].

These issues are related to their black-box nature. *Black-box model* is a term for a model where the relation between input and output is unknown. Machine learning models are especially prone to be classified as such, because physical models are replaced by linear regressions and general mathematical functions with no intrinsic meaning to the original problem, leading to a severe lack in interpretability. For this reason, machine learning models are notoriously poor at extrapolating to untrained loading conditions and are typically only useful to make approximations within the loading space of the training data.

The issues described above have been subject to extensive research over the last years and efforts to create more robust models have resulted in the development of various

different model that incorporate physics into their network. The reasoning being that the introduced physics force the network to comply the physical behaviour of the material, increasing interpretability and extrapolating abilities. Various methods have been proposed: *Physics Informed Neural Networks (PINN)* [45] introduce physics into the loss function, expanding the loss to three sub-units: the observed data loss, the physics loss and the boundary loss. *Thermodynamics-based Artificial Neural Networks (TANN)* [46, 47], use different sub-ANNs to predict increments of the internal state variables and the energy potential in order to solve for the dissipation rate and the stress increment. *Deep Material Networks (DMN)* [48], predict the properties of materials using mechanistic homogenization theory of RVEs. In the DMN, the nodes are redefined as *building blocks*, each of these building blocks represents a simple structure with analytical homogenization solutions. *HyperCAN* [49] utilises hypernetworks to construct adaptable constitutive artificial neural networks, which can be integrated into multiscale simulations of truss metamaterials. Each network offers its own advantages and characteristics. For this study, the most resemblance can be found in *Physically Recurrent Neural Networks (PRNN)* [50].

2.4.5 Physically Recurrent Neural Networks (PRNN)

Capturing history-dependent material behaviour while reducing the black-box behaviour of the model can be done through physically recurrent neural network (PRNN) modelling. The method is proposed by Maia et al. [50] and is used in combination with the FE² method, where neural networks are employed to accelerate the microscale predictions. The architecture of this model can be seen in Figure 2.13 and shows that the model contains three main elements. The first is the encoder, which converts macroscopic strains into local and fictitious strains. The second block is the material layer, introducing physics into the model. The third layer is the decoder, converting local and fictitious stresses back into macroscopic stresses.

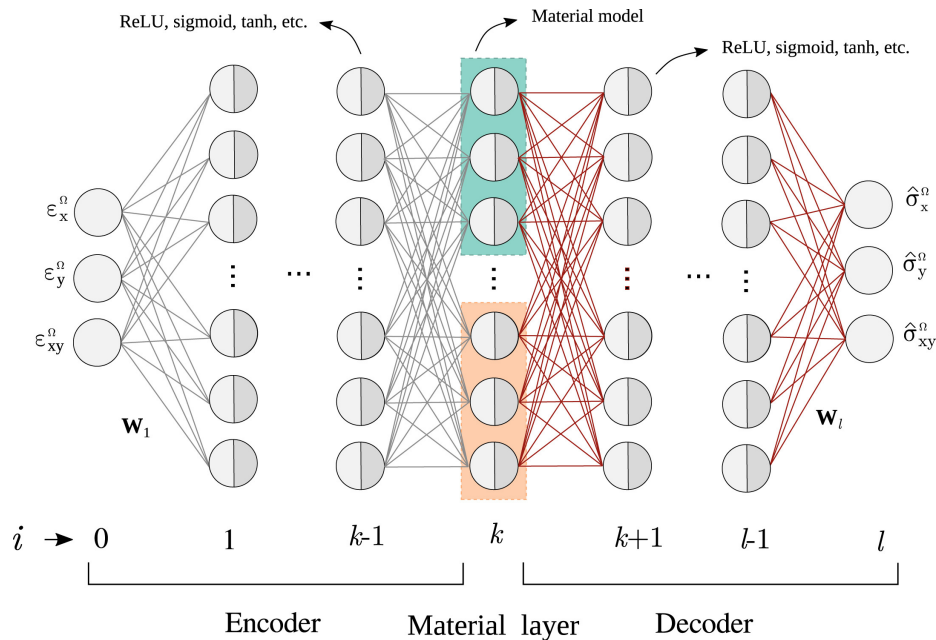


Figure 2.13: Physically Recurrent Neural Network architecture [50].

As mentioned, physics is introduced into the model through the material layer by grouping the strain components of each fictitious material point in a sub-group, shown in Fig-

ure 2.13 through differently coloured boxes, which will be evaluated individually. These sub-groups are passed through a material cell, containing the user-defined constitutive material model, converting the local strains to local stresses. In order to take into account path-dependency or the history of the model, the internal variables α of the previous time step of each material point are stored for the next time step and automatically updated once a new set of strains is computed. Constitutive model calculations thus depend on the strain components of the current time step and the internal variables of the previous time step:

$$\sigma, \alpha^t = \mathcal{D}(\varepsilon, \alpha^{t-1}), \quad (2.40)$$

where \mathcal{D} is the constitutive model.

The material cell can be represented as shown in Figure 2.14.

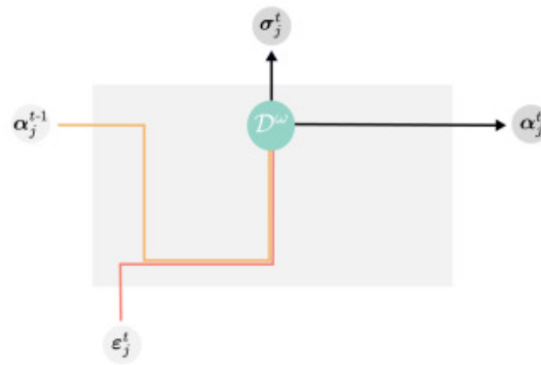


Figure 2.14: Material cell for PRNN [50].

PRNN are able to significantly outperform regular RNN models, while requiring significantly less training data, as the physics-based material models introduce built-in assumptions [50].

3

MODEL ARCHITECTURES

The purpose of this study is to extend the work by Maia et al. [50] on physically recurrent neural networks, described in Section 2.4.5, and propose a novel framework that reintroduces physics through beam theory. Instead of integrating a material layer to take material non-linearity into account, the beam neural network incorporates a beam-layer to take geometric non-linearity into consideration. First some preliminaries are discussed in Section 3.1 and the benchmark FNN is given elaboration in Section 3.2, the BNN architectures are discussed in Section 3.3.

3.1 PRELIMINARIES

In this section, training aspects such as data normalisation and choice of loss function are summarised. In the following, the network architectures investigated in this work are discussed.

Firstly, the stress-values of the datasets are normalised between -1 and 1 to improve training of the network. This is done separately for each stress-component:

$$\sigma_{j,norm} = \frac{2(\sigma_j - \sigma_{j,min})}{(\sigma_{j,max} - \sigma_{j,min})} - 1, \quad (3.1)$$

where σ_j is the stress component of the curve to be normalised and $\sigma_{j,min}$ and $\sigma_{j,max}$ are the minimum and maximum values for that stress component in the full training dataset. Secondly, to assess the performance of the network, a loss function is required. During training, the root mean squared error is used:

$$\mathcal{L} = \frac{\sqrt{\sum(\sigma_{True} - \sigma_{pred})^2}}{n_{data}}, \quad (3.2)$$

where \mathcal{L} is the loss, σ_{True} and σ_{pred} are the true and predicted values of the stress tensor and n_{data} is the number of data points within the curve, which is the number of curves in the batch, multiplied by the number of stress-components in each curve (three), multiplied by the number of loading steps per curve.

To evaluate the validation error of the system, the network output is first denormalised using σ_{min} and σ_{max} from the training data and then an error is computed using a relative root mean squared error (RRMSE) to get a percentage error of the network performance:

$$\mathcal{L}_{rel} = \sqrt{\frac{\sum(\sigma_{True} - \sigma_{Pred})^2}{\sum(\sigma_{True}^2)}}. \quad (3.3)$$

3.2 FEED-FORWARD NEURAL NETWORK (FNN)

The first network to be discussed is the most straightforward, namely a fully connected feed-forward neural network, shown in Figure 3.1. This network has three inputs: strains ϵ_{xx} , ϵ_{xy} and ϵ_{yy} , a single hidden layer and an output layer with three stress outputs:

σ_{xx} , σ_{xy} and σ_{yy} . Both the output and hidden layers have biases, but the nonlinear activation function is only considered in the former.

Three different activation functions that connect the input layer to the hidden layer are investigated: *ELU*, *tanh* and *softplus*.

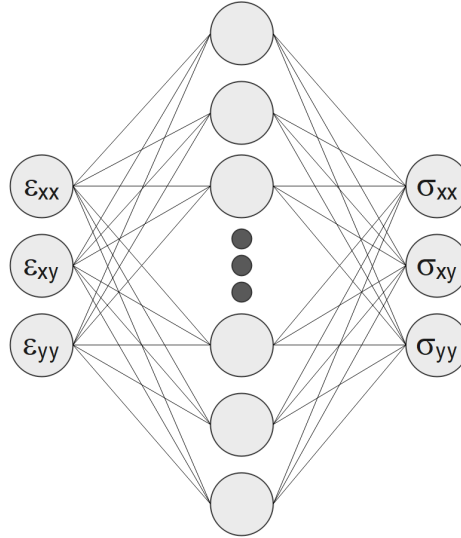


Figure 3.1: Feed-Forward Neural Network architecture.

3.3 BEAM NEURAL NETWORKS (BNN)

Next to be investigated is the Beam Neural Network (BNN), not to be confused with Bayesian Neural Networks. Three variants are proposed. Two utilise Euler-Bernoulli beam theory, of which one allows for a change in angle of the beams within the beam-layer (Section 3.3.2), and one that does not (Section 3.3.1). The former being able to fit non-linear data, the latter not. The third network makes use of the finite element method to model cantilever beams within so-called FE-boxes (Section 3.3.3).

3.3.1 Linear Beam Neural Network (BNN-L)

The general architecture of the Linear Beam Neural Network (BNN-L) is shown in Figure 3.2. This network can be subdivided into three components: the encoder, the beam-layer and the decoder. The encoder linearly translates the global input strains into sets of two local displacements, x and y . In turn, the beam-layer uses Euler-Bernoulli Equation 2.5 and Equation 2.6 to calculate the local forces, which are converted to local stresses by dividing by the cross-sectional area of the beam for tensile forces and the apparent top surface of the beam for bending forces. Given that the global output stresses are normalised, a normalisation is performed on the output of the beam-layer, based on the minimum and maximum values of stress components σ_{xx} and σ_{yy} in the training dataset. Having the nodal values of the beam-layer output in the same order of magnitude and the nodal values of the output layer allows for improved training of the network. Lastly, the decoder linearly translates the local stresses to the global stresses. Throughout the

network, no activation functions are used and bias is turned off to ensure that a zero-strain input results in a zero-stress output.

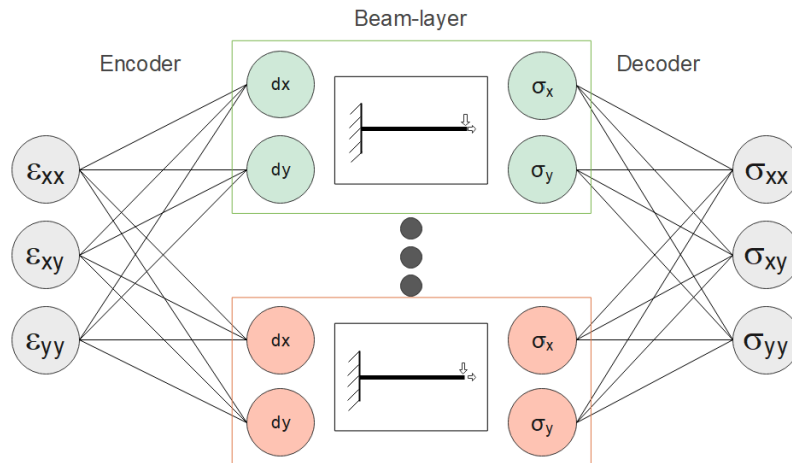


Figure 3.2: Linear Beam Neural Network (BNN-L) architecture.

Whereas the BNN-L in its basic form contains all-horizontal beams in the beam-layer, a variant investigated in this work consists of having beams with different initial angles. For a network with one beam, it is set horizontally, at 0 degrees. With two beams, one is set at 0 and the other at 45 degrees. Any network with more than two beams has these beams spaced evenly between 0 and 90 degrees. Besides comparing networks with and without different initial beam angles, the optimal number of beams within the beam-layer is set as a hyperparameter and will be investigated.

3.3.2 Non-Linear Beam Neural Network (BNN-NL)

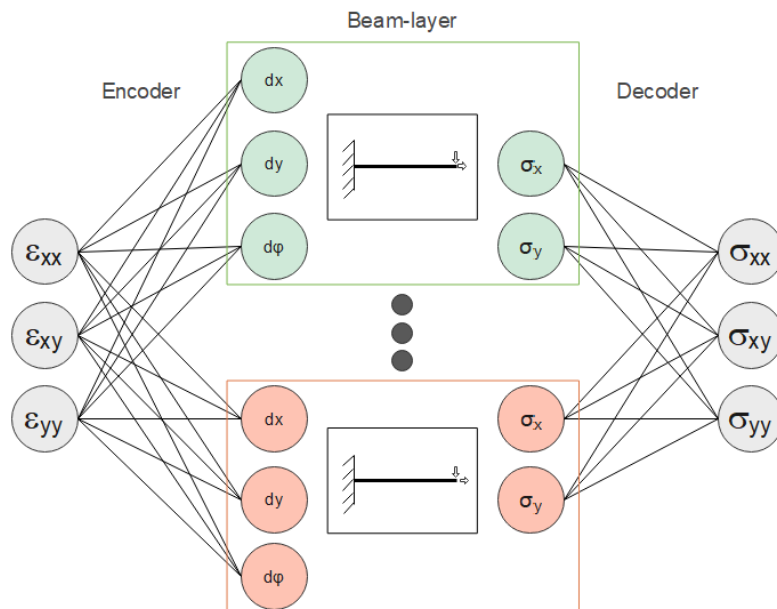


Figure 3.3: Non-Linear Beam Neural Network (BNN-NL) architecture.

Because the equations used in the beam-layer, Equations 2.5 and 2.6, contain a linear relation between stress and strain, no non-linearity is introduced into the network, given the absence of activation functions. Because of this, the linear beam neural network is

only able to predict linear outputs to, potentially, non-linear data. The Non-Linear Beam Neural Network (BNN-NL) aims to solve this. It is identical to the BNN-L, except that the encoder also outputs an angle increment, $d\phi$, to the beam-layer, illustrated in Figure 3.3. While the initial angle of the beams are defined a priori, as discussed in Section 3.3.1, $d\phi$ allows the network to change the angles of the beams with increased loading, introducing a non-linear source to the network.

Just like the BNN-L, the number of beams in the beam-layer is considered a hyperparameter to the system which will be investigated. Additionally, the effects of all-horizontal or variable initial beams angles will be explored.

3.3.3 Finite-Element Beam Neural Network (BNN-FE)

The third network investigated in this research is the Finite-Element Beam Neural Network (BNN-FE), illustrated in Figure 3.4. This network employs static, non-linear finite element models in the beam-layer, within so called *FE-boxes*, to introduce non-linearity into the network. Each FE-box contains an one-dimensional cantilever beam, consisting of 4 nodes, i.e. 3 elements. Each node has 3 degrees of freedom (DoFs), namely displacements dx and dy and rotation $d\phi$. For the node at the clamped side of the cantilever beam, all DoFs are constrained to zero. For the end-point node, the dx and dy displacements are constrained by the encoder output, which are, just like in previous networks, coupled in sub-groups of two. All other DoFs are free. The finite element model in each box is then solved through the *Iterative solving scheme* outlined in Algorithm 1 in Section 2.3.1.2, using the supplementary equations of Section 2.3. In this study, linear shape functions are used to approximate the displacements and solving the system of equations is done through Gauss integration. Shear locking is prevented by underintegration, using only one integration point in the middle of each element.

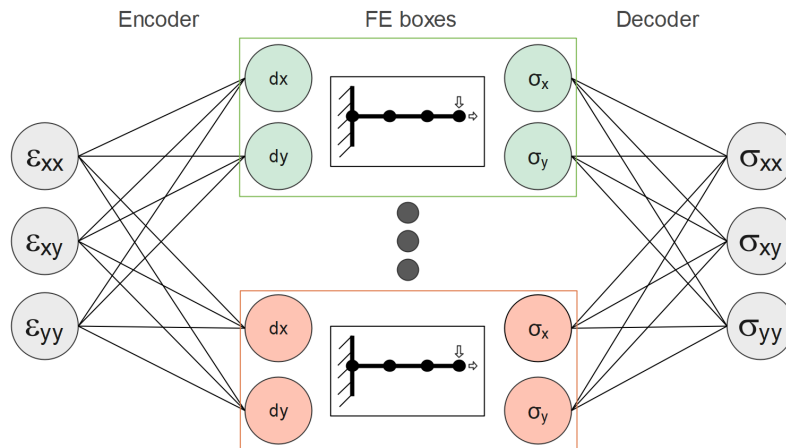


Figure 3.4: Finite-Element Beam Neural Network (BNN-FE) architecture.

Unlike the BBN-L and BNN-NL networks (Sections 3.3.1 and 3.3.2), for reasons that will become clear in Chapter 6, the cantilevers in the beam-layer are analysed at fixed, horizontal positions, that are unable to change their angle with loading increments. Also, no additional activation functions or biases are applied.

A notable feature of the BNN-FE is the relatively easy incorporation of dynamics into the network, as the state of the beams is known for every time step. This could become

important in future research, where BNN-FE could become an alternative to RNNs which suffer from the curse of dimensionality, where the BNN-FE does not.

An important thing to note, however, is that an implication of using finite elements in the beam-layer is that computational effort is greatly increased. Therefore, in order to reduce the time-requirements and thus increasing the number of training curves in the analyses, two changes have been made. Firstly, the assembly of the stiffness matrix \mathbf{K} and internal force vector \mathbf{f}_{int} have been vectorized, such that the system of equations can be solved for multiple curves and multiple FE-boxes at the same time. Secondly, even though the original dataset contains 150 loading steps per curve, the dataset for analysing the BNN-FE contains 11 loading steps, equally spaced out along the original 150. Although this causes an increase in the iteration steps required to converge the Newton-Raphson scheme, time requirements are greatly reduced overall without losing accuracy. Lastly, the initial ability of the Newton-Raphson scheme to converge towards a solution greatly depends on the initial displacement assignments of the encoder. In order to avoid non-convergence of the scheme due to a too large difference between the solution and the initial guess of the solution, the initialisation of the weight parameters of the encoder are modified to a narrower range where displacements are expected to be smaller.

4

DATA GENERATION

In this study two different datasets are used, one being for a honeycomb lattice architecture (shown in Figure 4.1), the other for a re-entrant lattice architecture (shown in Figure 4.2). Using these two datasets allows for a comparison of the network performance between an isotropic material: a hexagonal honeycomb (in linearity), and an auxetic material, the re-entrant structure. The data used for training, validating and testing the models in this work are obtained using a full-field finite element simulation of a single (1x1) unit cell of a 2D lattice structure with periodic boundary conditions. During the simulation, contact forces between beams are not taken into account. Results along the border of the unit cell are homogenized to obtain the displacement gradients and Piola-Kirchhoff stresses, relating the stresses to an undeformed reference configuration. Lastly, the displacement gradients H_{11} , H_{12} , H_{21} , and H_{22} are converted to strain to complete the datasets using the following 2D relations [51]:

$$\epsilon_{xx} = H_{11} \quad (4.1)$$

$$\epsilon_{xy} = \frac{H_{12} + H_{21}}{2} \quad (4.2)$$

$$\epsilon_{yy} = H_{22}. \quad (4.3)$$

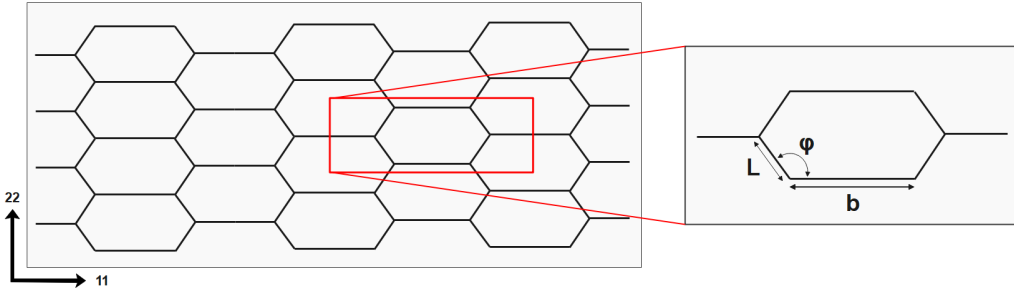


Figure 4.1: Schematic of honeycomb lattice.

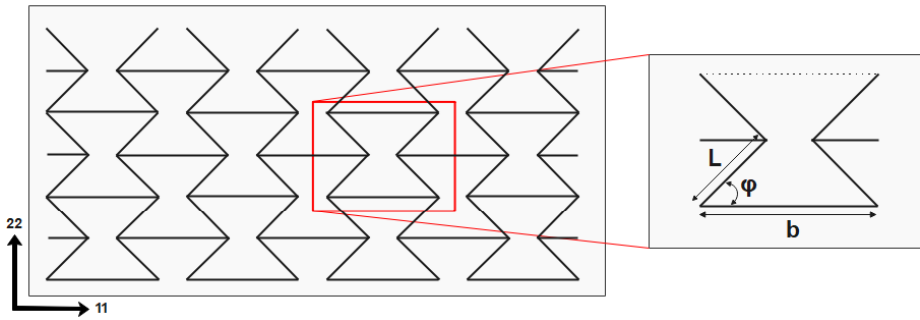


Figure 4.2: Schematic of re-entrant lattice.

To ensure an equal amount of normalised strain for each curve in the dataset, a unit sphere is placed in the loading space, on the surface of this sphere the loading conditions are generated. Because of this, every curve in both the honeycomb and the re-entrant datasets have equal magnitudes for the resultant strain vectors, namely 0.15. This ultimate strain is achieved within the dataset through 150 loading increments for which the strain and stress values are recorded. A visualization of this is shown in Figure 4.3 for a random

selection of 100 curves. In total, both the honeycomb and the re-entrant datasets contain the stress-strain curves for 10.000 different loading cases.

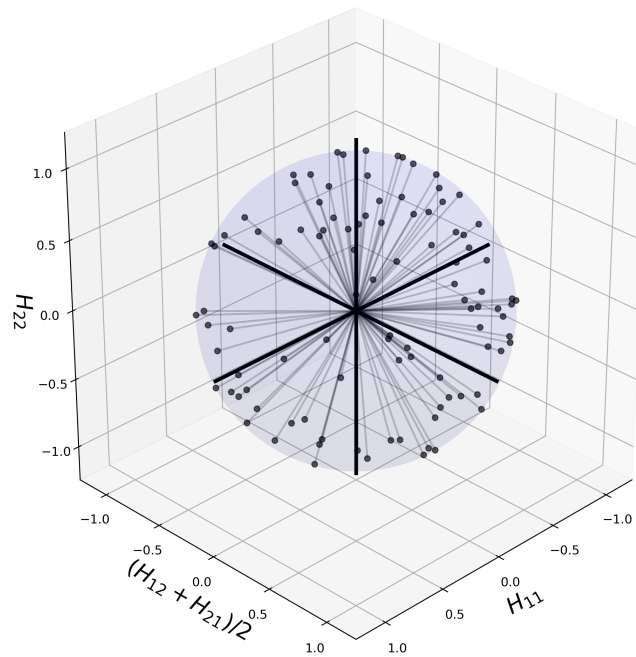


Figure 4.3: Unit-spherical loading space for data generation.

To avoid floating point errors, the base length of both lattices (length b in the schematics of Figures 4.2 and 4.1) are set to 1 meter. The dimensions of both lattice cells are shown in Table 4.1, where it can be seen that the only different between the lattices is the angle φ . The material properties are constant, meaning that no material non-linearity is observed, only geometric non-linearity.

Table 4.1: Material specifications.

Material properties						
Young's modulus E :	210 GPa	Poisson Ratio ν :	0.3			
Unit cell dimensions				Honeycomb	Re-entrant	
Strut length L :	0.5 m	Strut length b :	1 m	Angle φ :	120 deg	23.86 deg
Strut dimensions						
Cross-section:	Circular	Radius:	0.05 m			

5

MODEL SELECTION AND EVALUATION

To determine the optimal network sizes and in order to compare network performance, a selection procedure is employed. This section covers the choices made in this study.

5.1 DATASET SIZES

As outlined in Chapter 4, a total of 10,000 curves were obtained per material architecture. The datasets are split into a training set and a validation set, where the training set is used to train the network and the validation set is used to determine the network performance on unseen data. It is desirable to use as much training and validation data as possible, while maintaining reasonable time-requirements for the training of the network.

During training, the validation set remains the same set of curves throughout every epoch, while the training dataset might vary if less curves are used than are available. This process is shown in Figure 5.1, where the curves that are used during the epoch are symbolised in red. It is shown that the validation set is always the same set of curves, while the training curves are picked at random and used throughout the remainder of the training. Note, however, that in case the training set contain all available curves, no random selecting of curves is performed.

Throughout this study, the number of curves in the training and validation datasets will be reported, as this may vary per network and analysis. Networks that share identical validation datasets may be compared directly. For networks where this is not the case, a test dataset must be used, against which both networks can be fairly compared.

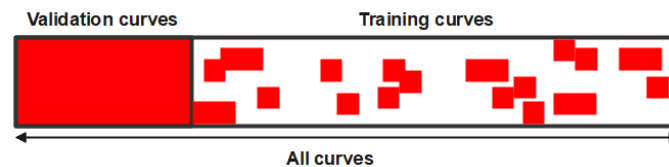


Figure 5.1: Visualisation of fixed curves in validation dataset, but randomly selected curves in training dataset.

5.2 HYPERPARAMETER CHARACTERISATION

The hyperparameters to be investigated in this work have been discussed in Chapter 3 and are summarised in Table 5.1.

Table 5.1: Hyperparameters to be investigated per network.

Networks	Hyperparameters
FNN	Activation function Number of nodes in hidden layer
BNN-L / BNN-NL	Number of beams in beam-layer
BNN-FE	Number of FE-boxes in beam-layer

To test and compare hyperparameters, networks are trained for a range of possible values of the hyperparameter. While training, the validation error is checked every 50 epochs. The lower this validation error, the better the performance of the network. During training, the validation error tends to decrease at first, but after some epochs, overfitting of the training data may occur, where the validation error increases again. This is illustrated in Figure 5.2, where the best validation error is reached around epoch 550, whereafter overfitting occurs. Therefore, a criterion can be set, where training is stopped when no improvement to the validation error has been achieved for a number of epochs. In this work, this criterion is set to 150 epochs of no improvement. The model with the lowest validation error is then saved.

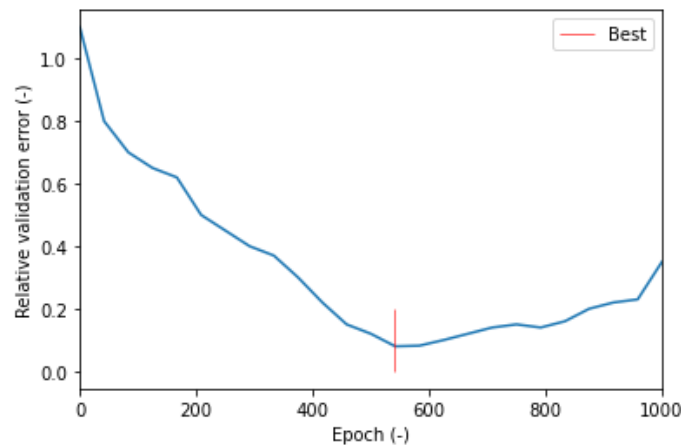


Figure 5.2: Validation error per epoch during network training.

5.3 LEARNING CURVES AND MODEL SELECTION

In order to compare different types of networks, their performances are assessed by varying the number of training curves and calculating their validation errors. Besides aiming for the lowest validation error, it is desirable for a network to require the least amount of training data to converge towards its best performance. It is then up to the user to decide what properties of the network are prioritised and what network is deemed the *best*.

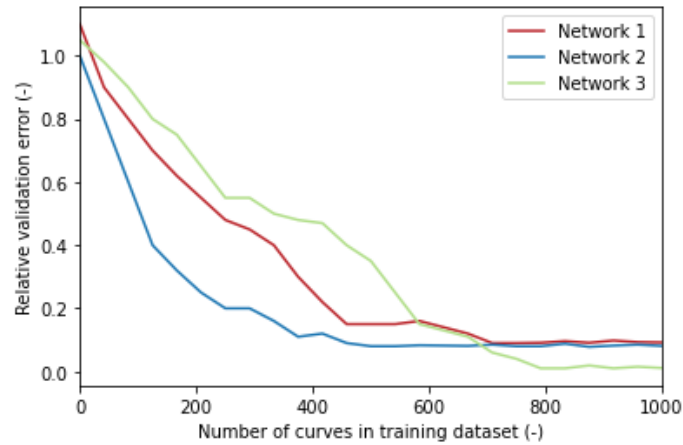


Figure 5.3: Learning curve for three networks using mock data.

An example of such a *learning curve* is given in Figure 5.3, where three different networks are depicted. In this case, network 2 converges quickly towards a low validation error, while network 3 requires more training data, but converges towards a lower validation error overall. It is up to the user to decide whether or not the performance gain of network 3 is worth the higher training data requirement. Note that, as stated in Section 5.1, the validation datasets of each network need to be identical in order to compare them fairly. If this is not the case, network performance must be assessed using a test dataset.

Additionally, to assess the impact of incorporating physics into the networks, they are evaluated in extrapolation. For this, networks are evaluated on a test dataset that contains more loading iterations than the training data. For each iteration step, the error between the network's prediction and the true value is calculated. This is repeated over a large number of test curves, which allows for the plotting of the average performance of a network along a range of iteration points, both for interpolation and extrapolation. An example plot is shown in Figure 5.4. It shows that while networks may perform well in the training range, their behaviour in extrapolation might be very different. Network 1 performs best in the training range, while network 2 performs better in extrapolation. Given that the aim of this study is to make a more robust model, better capable of complying with the physical behaviour of the material, network 2 would be preferred. Overall, a reasonable target in this study would be to keep the prediction error within a 10% margin, somewhat comparable to the macroscale simulation error.

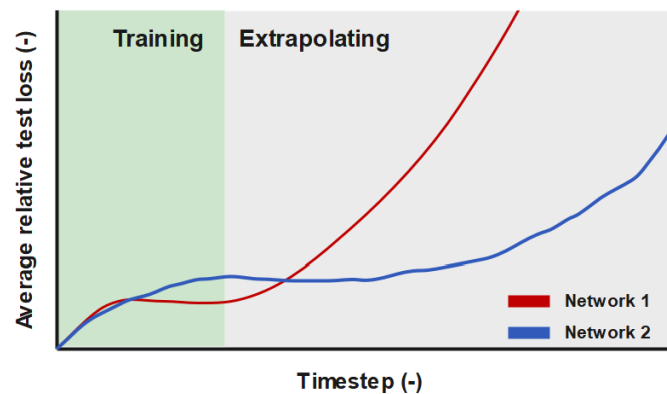


Figure 5.4: Extrapolation behaviour of two networks using mock data.

6

RESULTS AND DISCUSSION

In this section, results are summarised. Section 6.1 outlines the choices made for the hyperparameter characterisation. Section 6.2 compares the performance of the investigated networks in the training range, while Section 6.3 assesses the networks in extrapolation. Additionally, some exploratory studies have been done, that build upon the findings of the previous sections, these are provided in Section 6.4.

Throughout this chapter, results are obtained by averaging network outputs over multiple runs, for this study averages of 15 runs are used. The shaded areas in plots show the boundaries of the minimum and maximum results. Additionally, all networks utilise a batch size of 4 and an initial learning rate of 0.01, except for the BNN-FE that requires an initial learning rate of 0.001 in order to avoid non-convergence of the Newton-Raphson scheme.

6.1 HYPERPARAMETERS

As discussed in Chapter 3 and summarised in Table 5.1, each network has hyperparameters that require evaluation. This section covers this process.

A shared investigation into BNNs and the effect of the initial angles of the beam in the beam-layer has been provided in Appendix A. Based on that study, it is verified that there is no effect of the initial angles of the beams in the beam-layer on the performance of the networks. From this point onwards, all the beams in the beam-layer have a set initial angle of zero degrees (a horizontal cantilever). An overview of the number of curves per dataset used for the hyperparameter characterisation has been provided in Table 6.1. Given larger computational effort requirements for the BNN-FE, a reduction of curves in the training and validation datasets had to be imposed, as well as the data being sparse instead of dense, i.e. the curve is covered by 11 data points instead of 150.

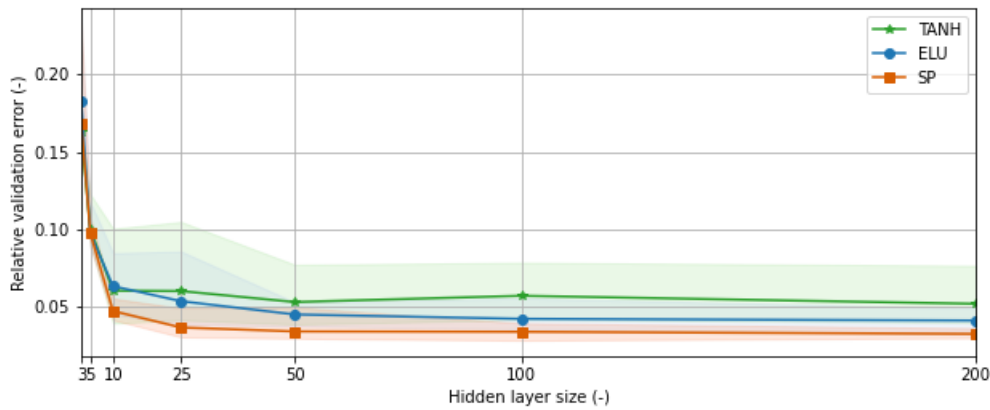
Table 6.1: Number of curves per dataset for hyperparameter characterisation, for both honeycomb and re-entrant lattices.

	Training	Validation
FNN	8000	2000
BNN-L	8000	2000
BNN-NL	8000	2000
BNN-FE	500	500

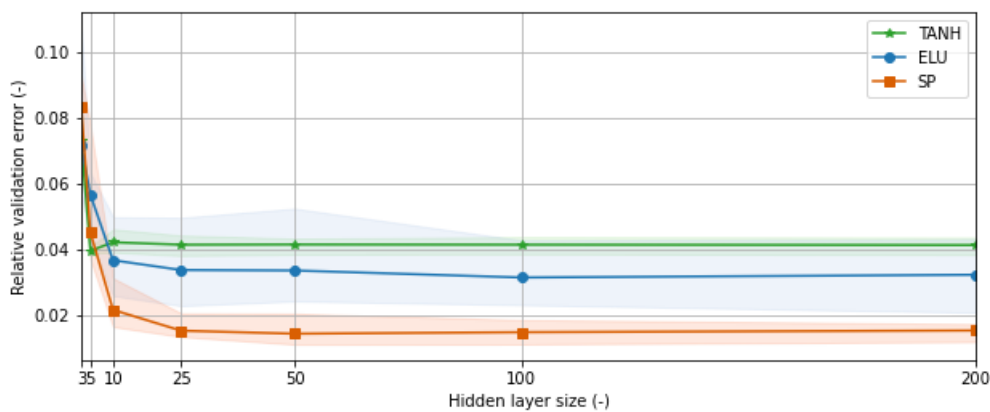
Feed-Forward Neural Network (FNN)

For the FNN, the hyperparameters that are investigated are the number of nodes in the hidden layer and the activation function that connects the input layer to the hidden layer. As a reminder, the activation functions to be investigated are the *ELU*, *tanh* and *softplus*

functions, graphically illustrated in Figure 2.10. Results of the FNN hyperparameter characterisation are shown in Figure 6.1.



(a) Honeycomb



(b) Re-entrant

Figure 6.1: FNN hyperparameter characterisation. Relative validation error per hidden layer size, for different activation functions. For a honeycomb lattice (a) and a re-entrant lattice (b).

It can be seen that for both lattice architectures the softplus activation function performs best. Moreover, for neither architecture any performance is gained from increasing the hidden layer size above 25 nodes. Therefore, the choice is made for the softplus activation function and a hidden layer size of 25 nodes.

Linear Beam Neural Network (BNN-L)

Next to be discussed is the BNN-L, where the hyperparameter to be investigated is the number of beams in the beam-layer. Results can be seen in Figure 6.2. It is clear that for both lattices only 2 beams are needed in the beam-layer to achieve convergence.

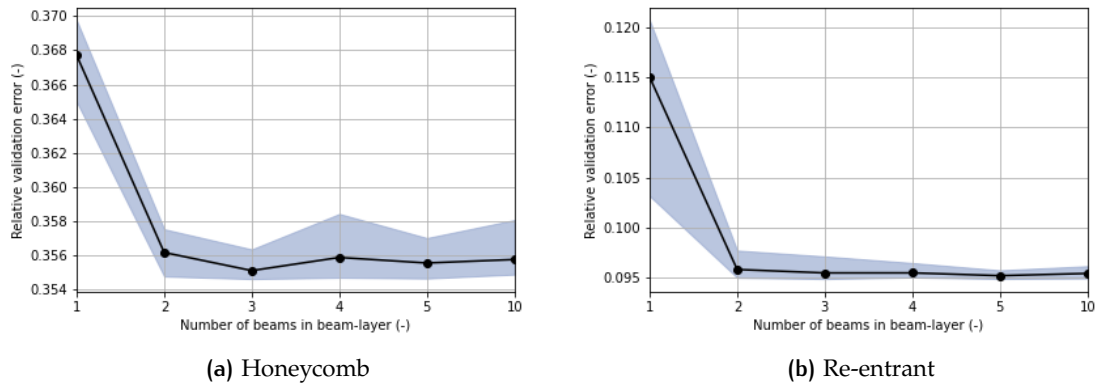


Figure 6.2: BNN-L hyperparameter characterisation. Relative validation error per number of beams in beam-layer. For a honeycomb lattice (a) and a re-entrant lattice (b).

Non-Linear Beam Neural Network (BNN-NL)

The BNN-NL, similarly to its linear counterpart, also requires investigating into the number of beams in the beam-layer to achieve convergence. The results are given in Figure 6.3 and show that for both lattice architectures 3 beams in the beam-layer is sufficient.

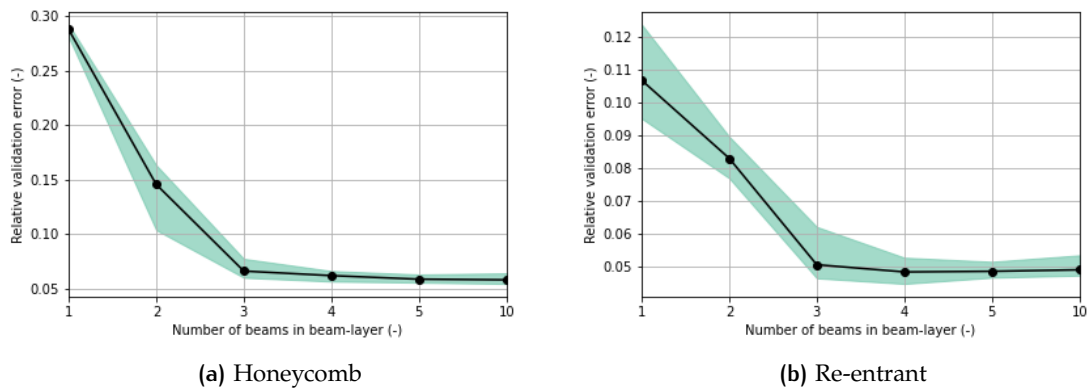


Figure 6.3: BNN-NL hyperparameter characterisation. Relative validation error per number of beams in beam-layer. For a honeycomb lattice (a) and a re-entrant lattice (b).

Finite Element Beam Neural Network (BNN-FE)

Lastly, the number of FE-boxes in the beam-layer for the BNN-FE needs to be determined. In a similar fashion, the results are shown in Figure 6.4, which show that the honeycomb lattice requires 4 FE-boxes in the beam-layer and the re-entrant lattice requires 5 FE-boxes in the beam-layer in order to have optimal performance.

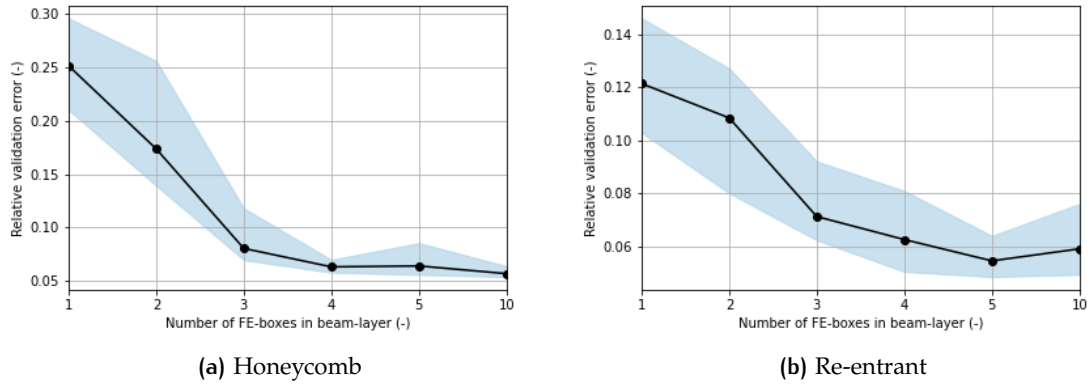


Figure 6.4: BNN-FE hyperparameter characterisation. Relative validation error per number of FE-boxes in beam-layer. For a honeycomb lattice (a) and a re-entrant lattice (b).

A summary of the choices on the hyperparameters of each network and both lattices is shown in Table 6.2.

Table 6.2: Hyperparameter choices per network, per lattice architecture.

Hyperparameters		Honeycomb choice	Re-entrant choice
FNN	Activation function	Softplus	Softplus
	Number of nodes in hidden layer	25	25
BNN-L	Number of beams in beam-layer	2	2
BNN-NL	Number of beams in beam-layer	3	3
BNN-FE	Number of FE-boxes in beam-layer	4	5

6.2 NETWORK LEARNING

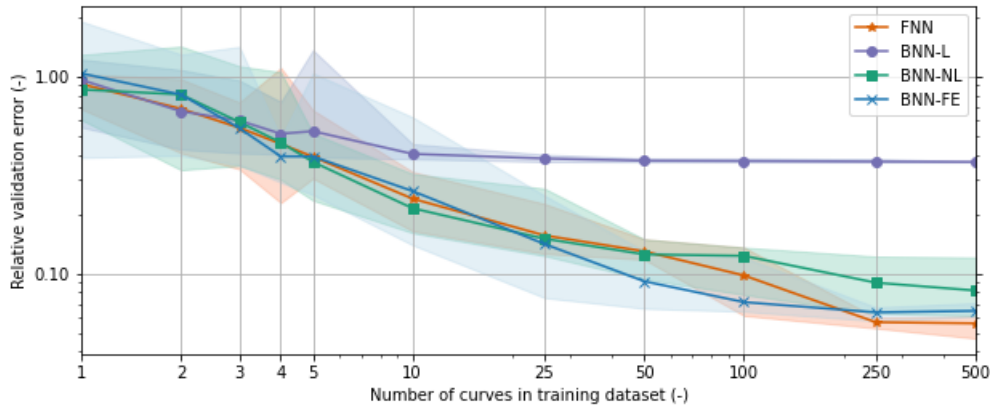
In order to compare networks and their ability to generalise, a learning curve can be generated. In a learning curve, the validation error is plotted against the number of curves in the training dataset, showcasing the performance of a network with different amounts of data, with emphasis on the limited data regime, also explained in Section 5.3.

Given that the BNN-FE required sparse data due to training time constraints, it is also worth looking at using sparse data for the other networks instead of dense data. The result is shown in Appendix B and confirms that there is no difference in network performance between sparse and dense datasets, therefore, from this point onwards sparse data is used. Moreover, in order to fairly compare the BNN-FE and the other networks, the validation datasets of the FNN, BNN-L and BNN-NL networks are adjusted to contain 500 curves. An updated table is shown in Table 6.3.

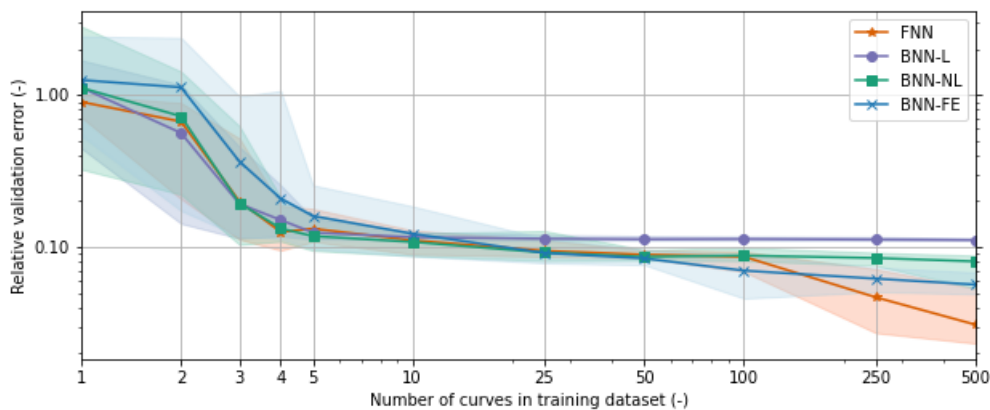
Table 6.3: Number of curves per dataset for network learning, for both honeycomb and re-entrant lattices.

	Training	Validation
FNN	Variable	500
BNN-L	Variable	500
BNN-NL	Variable	500
BNN-FE	Variable	500

The learning curves for both lattices are shown in Figure 6.5 and show that the learning efficiency of all networks are reasonably comparable, indicating that the physics integrated into the BNNs had no effect on the learning efficiency. Additionally, it can be seen that the linear predictions of the BNN-L are insufficient for predicting non-linear material behaviour, more so for the honeycomb lattice than the re-entrant lattice. The other networks show similar accuracy, although the FNN seems to have an edge over the BNN-FE and BNN-NL respectively. Which is unsurprising, given that a FNN is very capable of making accurate predictions withing the training range, given enough training data.



(a) Honeycomb



(b) Re-entrant

Figure 6.5: Learning curve. For a honeycomb lattice (a) and a re-entrant lattice (b).

In Figure 6.5 the FNN for both lattices and the BNN-NL for the honeycomb lattice do not show convergence yet. For this reason, extended learning curves can be found in Appendix C. Combining Appendix C with the results of Figure 6.5, the minimum number

of curves in the training dataset to achieve convergence can be determined, the result is shown in Table 6.4.

Table 6.4: Minimum number of curves per dataset required for network convergence.

	Honeycomb choice	Re-entrant choice
FNN	8000	2000
BNN-L	10	10
BNN-NL	2000	25
BNN-FE	100	100

6.3 EXTRAPOLATION

Where previous results have been focused on interpolation, assessment of the effects of the incorporated physics into the networks is best performed in extrapolation. This section will discuss the extrapolation results for the honeycomb lattice, whereas the results for the re-entrant lattice are provided in Appendix D to avoid repetition.

In order to assess networks in extrapolation, test datasets containing 400 (dense) curves are created that reach a normalised strain three times higher than in training, each with 450 time steps. The relative test error is then calculated per data point and plotted to graphically show the performance of each network, previously explained in Section 5.3. The average performance on the test dataset for all four networks can be found in Figure 6.6, where a clear distinction is made between trained and the extrapolated ranges.

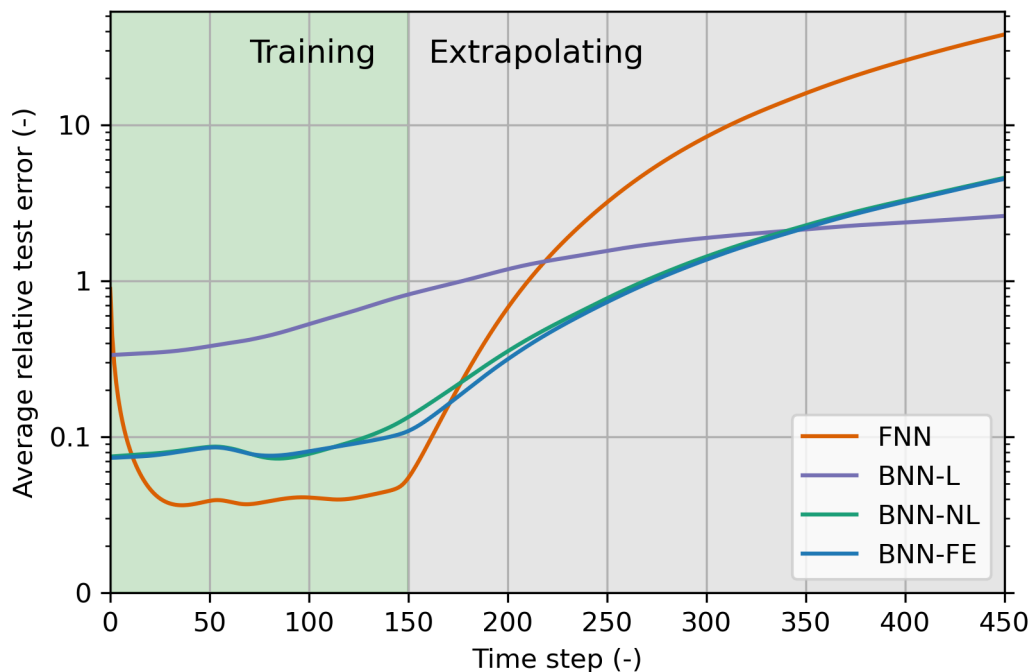


Figure 6.6: Performances of networks in extrapolation: honeycomb lattice.

A few interesting remarks can be made about Figure 6.6. The results confirm that the FNN shows strong performance within the interpolation range, while in extrapolation

this degrades rapidly, highlighting the black-box nature of the FNN. Moreover, the FNN shows disproportionately high relative error in early time steps, where stress values are relatively small. This is due to the inclusion of the bias term, resulting in large relative errors, even though absolute errors might be minor. Furthermore, both the BNN-NL and BNN-FE perform almost identically in both interpolation and extrapolation, showing that the integration of physics has enhanced its generalisation capabilities. Despite this, the accuracy of the BNN-L within the interpolation range is poor, indicating that linear predictions of non-linear material behaviour is insufficient. Nevertheless, the average performances of evaluated networks on the test dataset indicate that none demonstrate the required performance to be used in actual FE^2 modelling, failing to generalise well enough in extrapolation.

While the average performance on the curves in the test dataset might give some insight on the overall behaviour of the network, it fails to give information on the consistency of the networks, i.e. the spread of the results. Figure 6.7 aims to provide more insight into this, as it shows the performance for each of the 400 test curves.

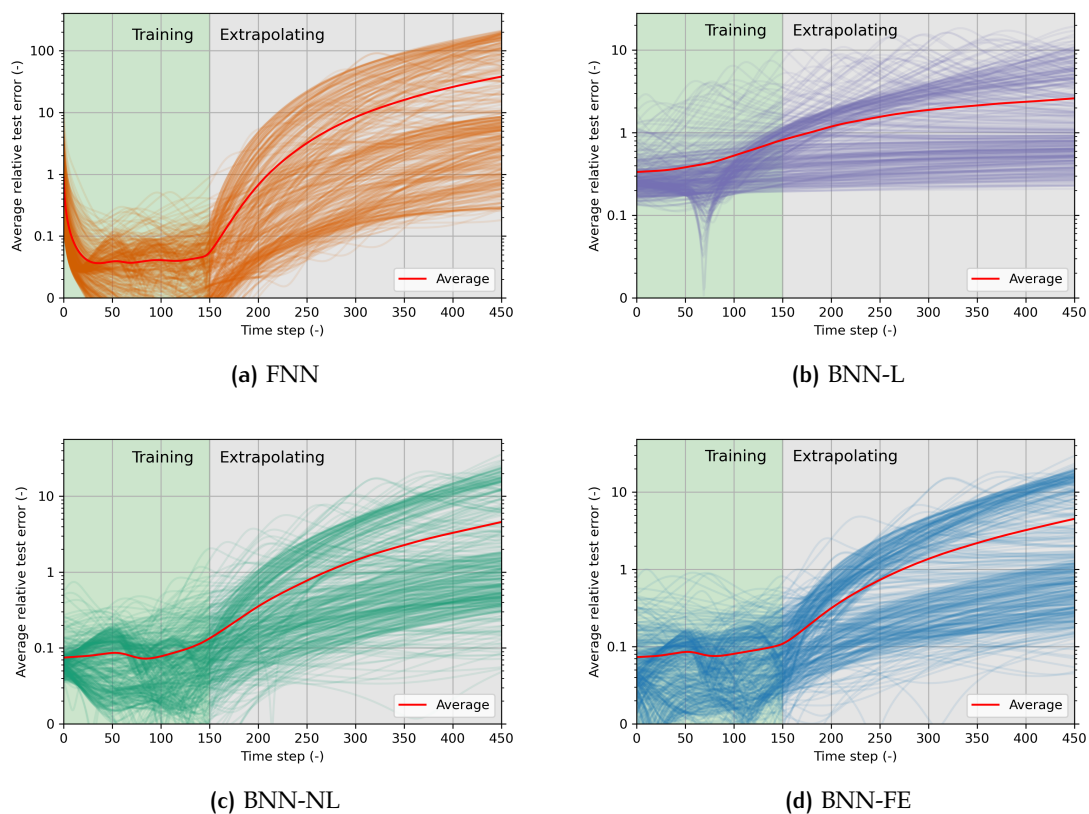


Figure 6.7: Honeycomb extrapolation, all 400 test dataset curves plotted per network.

It is revealed that there is a large spread in the results and that there are distinct paths that curves tend to follow. To explore this further, a plot can be generated where the loading directions are plotted with their corresponding relative test error. Figure 6.8a shows this for the BNN-FE network at time step 200, inside the extrapolation range, where the colours represent a discrete interval in the model's performance, from dark green (< 0.1 relative error) to dark red (> 1 relative error). Similar figures for the other models and at more time steps are included in Appendix E. Side views of Figure 6.8a are provided in Figures 6.8b, 6.8c and 6.8d to improve readability.

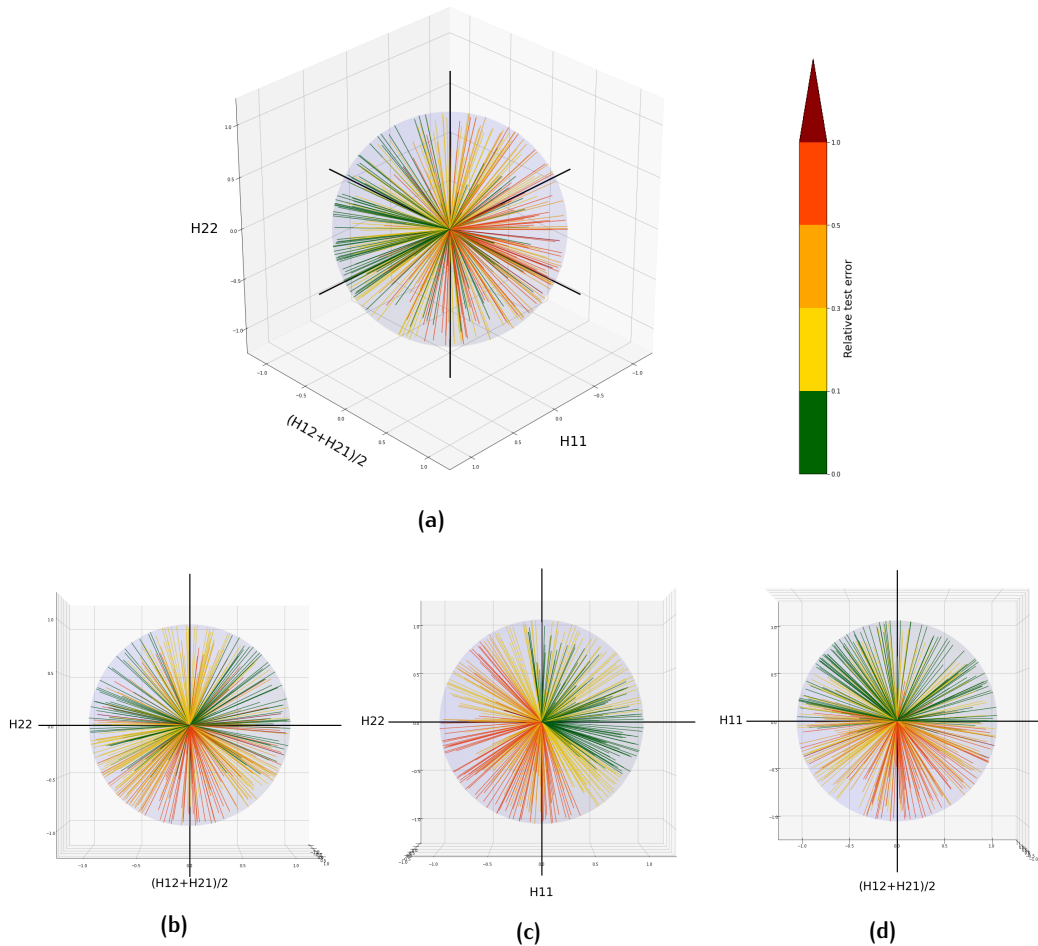


Figure 6.8: Honeycomb extrapolation performance in the loading space, BNN-FE at time step 200.

Figure 6.8 shows that there are indeed distinct regions of poorer performance, meaning that the model experiences more trouble in some loading conditions than in others. For the honeycomb lattice, this region of poorest performance is for combined compression in 11 and 22 directions. This is not a model specific issue, the same can be found for the other networks (FNN, BNN-L and BNN-NL), as shown in Appendix E. To confirm the intuition that the distinct grouping of curves in Figure 6.7 can be correlated to their positions in the loading space, the BNN-FE result in Figure 6.7d can be repeated, although with colours representing the curve's position in the loading space. In this case, the loading space is subdivided in four sections, where shear is excluded. The result is shown in Figure 6.9, confirming the correlation.

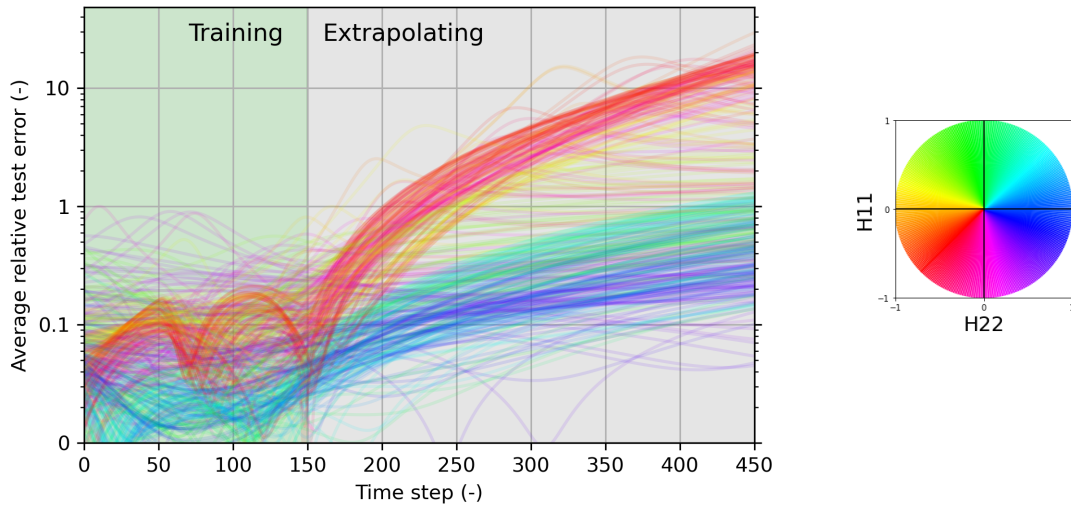


Figure 6.9: Honeycomb extrapolation, BNN-FE with colours representing the curves' position in loading space.

To find the reason for the distinct split performance, the network predictions for a curve in the region of poorer performance can be plotted (Figure 6.10) and combined with plots showing the movements of the beams in the beam-layer in comparison with the true deformations of the lattice, shown in Figure 6.11 for the BNN-FE. Beam-layer deformations for BNN-L and BNN-NL can be found in Appendix G and F respectively, where it can be seen that the Euler-Bernoulli assumptions of infinitesimal displacements are significantly stretched, extending beyond their original scope. The same can be done plotting a curve that shows relatively good performance: Figures 6.12 and 6.13. Also for this case, the beam deformations of the beams in the beam-layers of the BNN-L and BNN-NL are provided in Appendix G and F. The loading cases for these examples are shown in Table 6.5, the loading case representing the region of poor performance will be known as *loading case 1*, whereas the case representing the region of better performance will be known as *loading case 2*.

Table 6.5: Loading cases: honeycomb lattice.

	H11	(H12+H21)/2	H22
Loading case 1 (poor extr.)	-0.7500	-0.4330	-0.5000
Loading case 2 (good extr.)	0.5000	0.7071	-0.5000

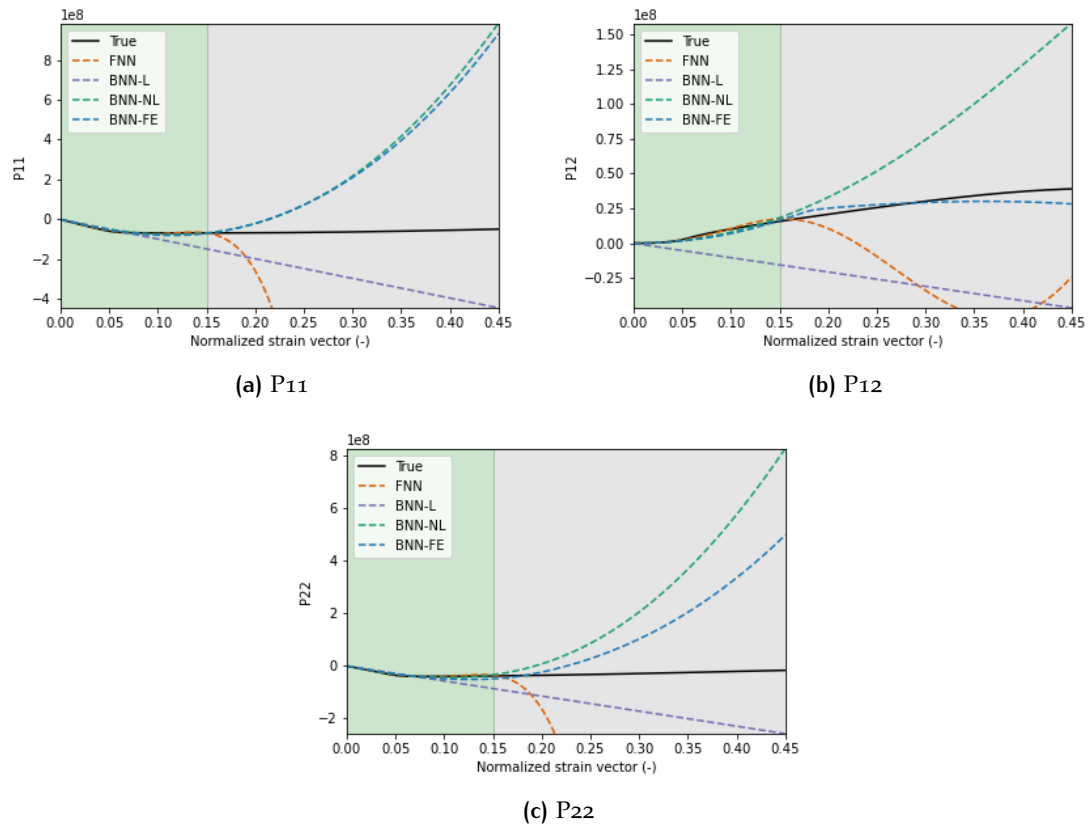


Figure 6.10: Honeycomb extrapolation predictions per network: loading case 1.

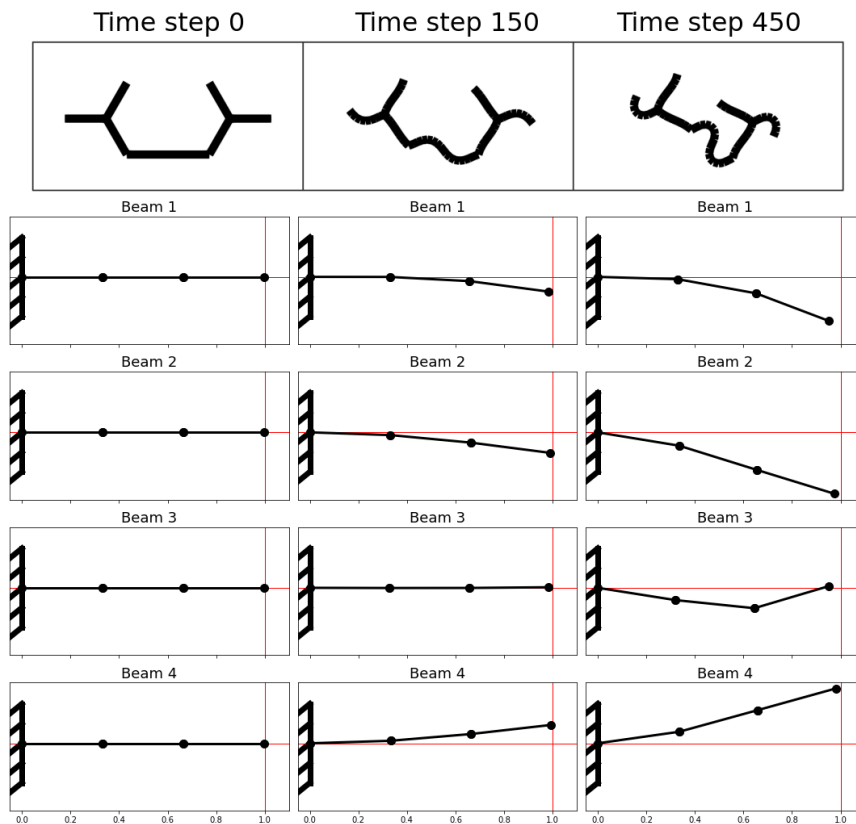


Figure 6.11: True lattice deformations and beam deformations in beam-layer of BNN-FE at time steps 0, 150 and 450 for a honeycomb lattice: loading case 1.

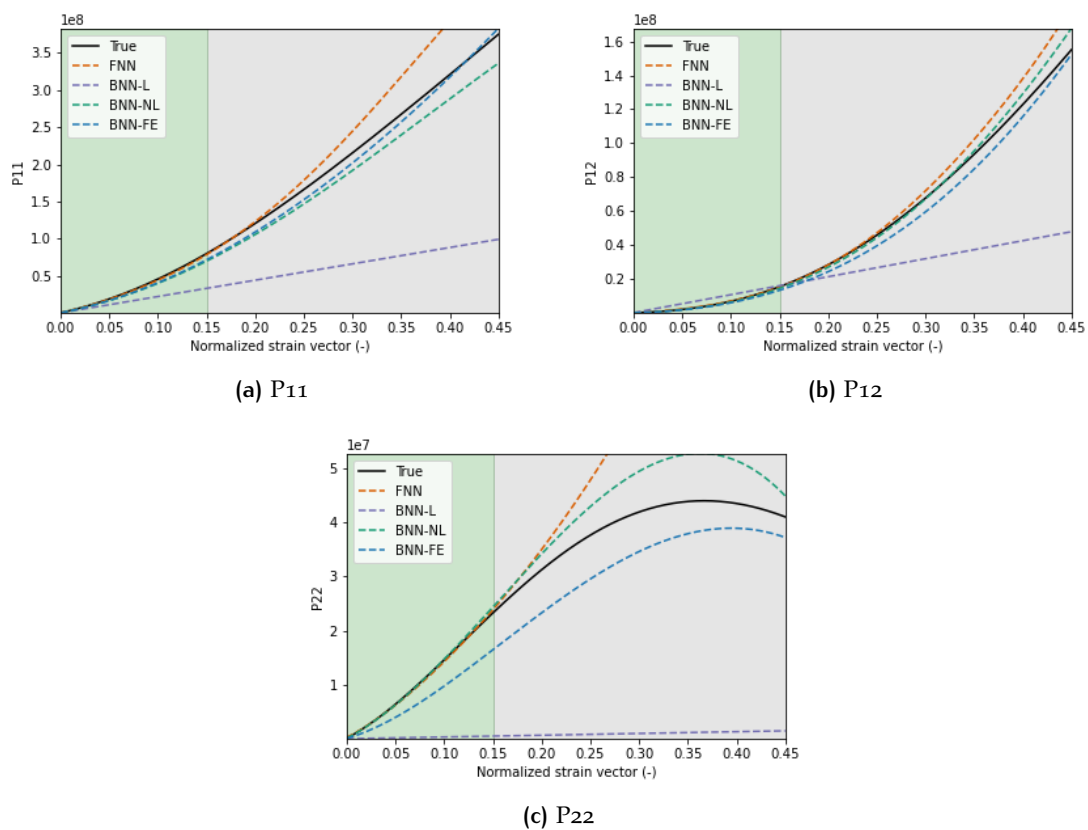


Figure 6.12: Honeycomb extrapolation predictions per network: loading case 2.

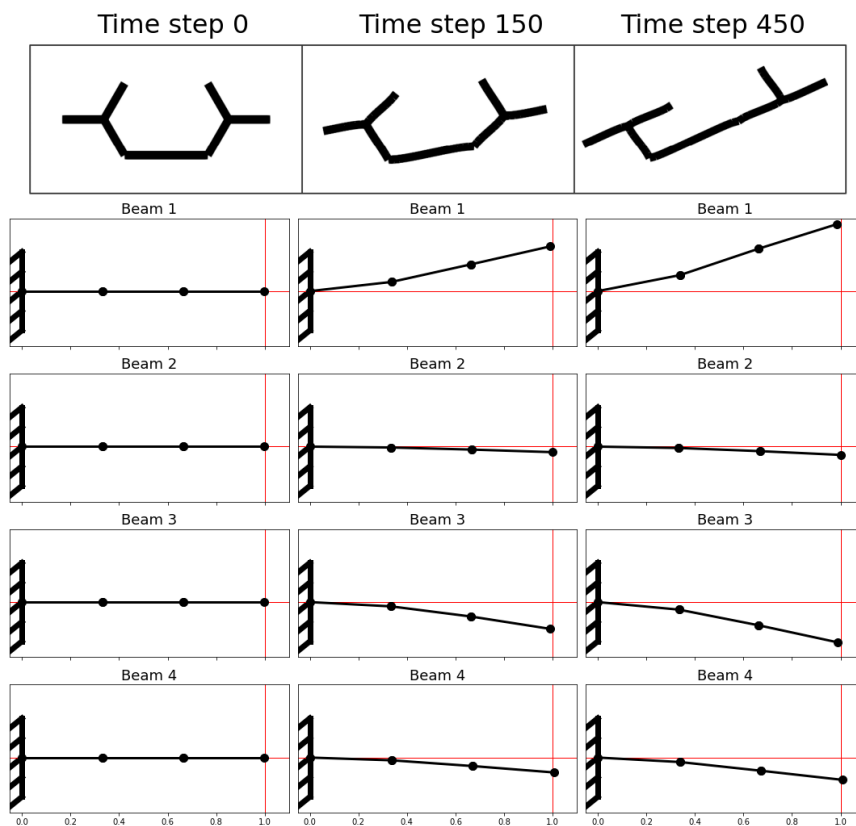


Figure 6.13: True lattice deformations and beam deformations in beam-layer for BNN-FE at time steps 0, 150 and 450 for a honeycomb lattice: loading case 2.

Comparing the lattice deformation in loading case 1 to the deformation in loading case 2, an immediate thing to note is the buckling behaviour due to the compressive loading in loading case 1. The beams in the beam-layer of the BNN-L and BNN-NL are inherently incapable of buckling due to limitations in Euler-Bernoulli beam theory, supported by the beam-layer deformations in Appendices G and F. The BNN-FE, however, is capable of modelling buckling and an effort to do so is shown in Figure 6.11, beam 3. However, the observed buckling deformation reveals a fundamental constraint to the current model: modelling buckling behaviour with only three elements leads to inaccurate representation of the physical response. Moreover, buckling of the lattice originates from beam-to-beam interactions, something that is not taken into consideration in the BNNs: more realistic buckling behaviour could therefore be achieved by interconnecting the movements of individual beams. This process highlights the benefit of interpretability in the model, something that is not possible for the FNN.

6.4 EXPLORATORY STUDIES

This section proposes improvements to the BNN-FE model to more accurately capture buckling behaviour, focusing only on the honeycomb lattice. While these proposals have been worked out, it is important to note that no formal model selection was conducted; instead, the parameters of the previously developed BNN-FE were directly adopted. As such, the findings presented here are only exploratory.

A first suggestion for improvement would be to increase the number of elements in the cantilever beam, adding more degrees of freedom to the model. Given that the full-field simulation used to generate the training data utilises 8 elements per beam in the lattice, an immediately obvious choice would be to mimic this in the FE-boxes, thus increasing the number of elements per beam from 3 to 8. A first attempt to do this is provided in Appendix H, showing no difference to the original 3-element BNN-FE. A second suggestion would be to introduce a non-linear encoder, giving the networks another source of non-linearity. In this work, to obtain a non-linear encoder, a hidden layer is inserted after the input layer, consisting of 100 nodes and equipped with the tanh activation function. From a physical point of view, adding a non-linear encoder would be advantageous, given that lattice displacements throughout loading are inherently non-linear. However, interpretability of the model is reduced, as it becomes more difficult to separate the effects of the non-linear encoder and what is being captured by the beams. Results are shown in Appendix I and, just like increasing the number of elements in the FE-model, show no real improvements from the original BNN-FE.

Behaviour changes, however, when the two previous suggestions are combined, resulting in a BNN-FE model with 8 elements in the FE-model and a non-linear encoder, illustrated in Figure 6.14.

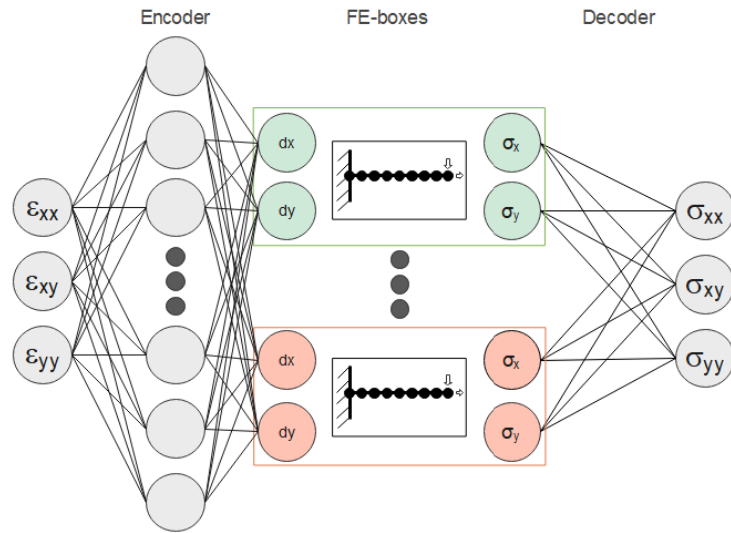


Figure 6.14: BNN-FE with non-linear encoder and increased number of elements in the FE-model.

The performance of the model in extrapolation is shown in Figure 6.15, where the average performance of the basic BNN-FE, that has 3 elements in the FE-model and a linear encoder, is included. For direct comparison, the same test dataset is used for both models. While each of the previous suggestions on their own were ineffective, the combination of the two reveals improvement.

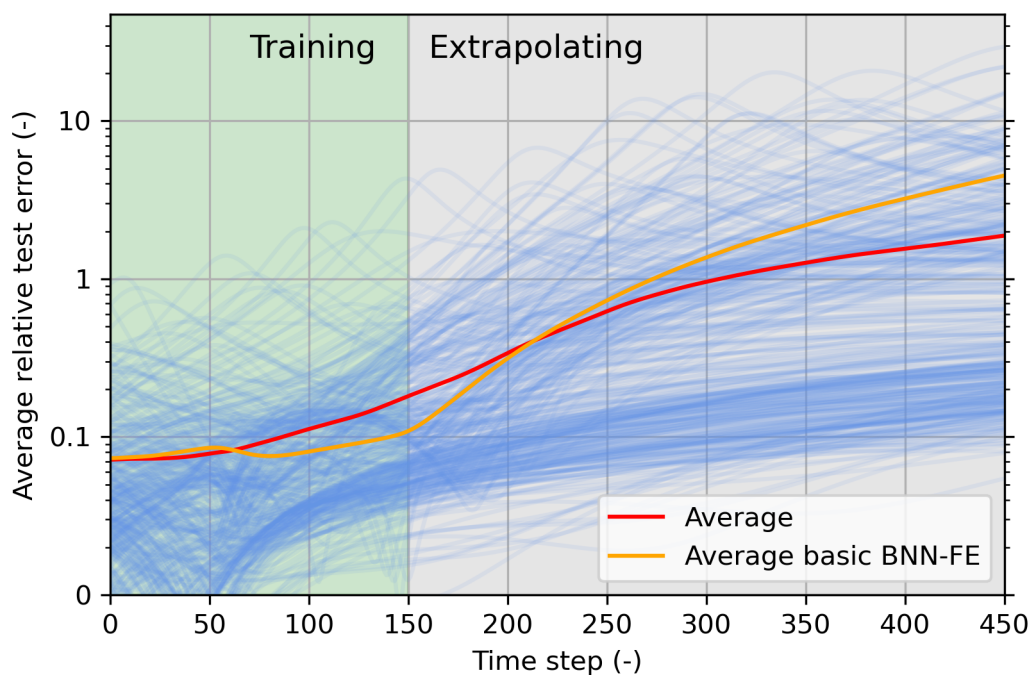


Figure 6.15: Performances of BNN-FE in extrapolation, with non-linear encoder and an increased number of elements in the FE-model: honeycomb lattice.

To identify the origin of the improvements, the performance of the model on each curve of the test dataset is plotted in the loading space. Figure 6.16 shows this for the network at time step 200, while more time steps are included in Appendix J. Additionally, the

deformations of the beams in the beam-layer, for loading case 2 of Table 6.5, are provided in Figure 6.17.

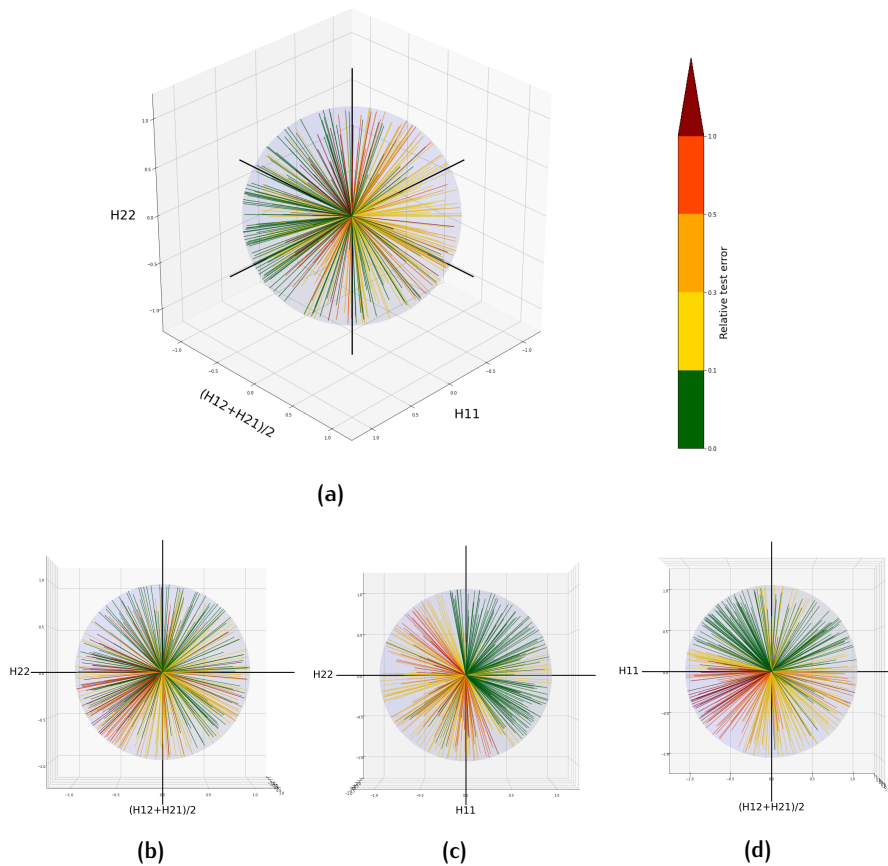


Figure 6.16: BNN-FE with 8 elements in FE-model and non-linear encoder: extrapolation performance in the loading space at time step 200, for a honeycomb lattice.

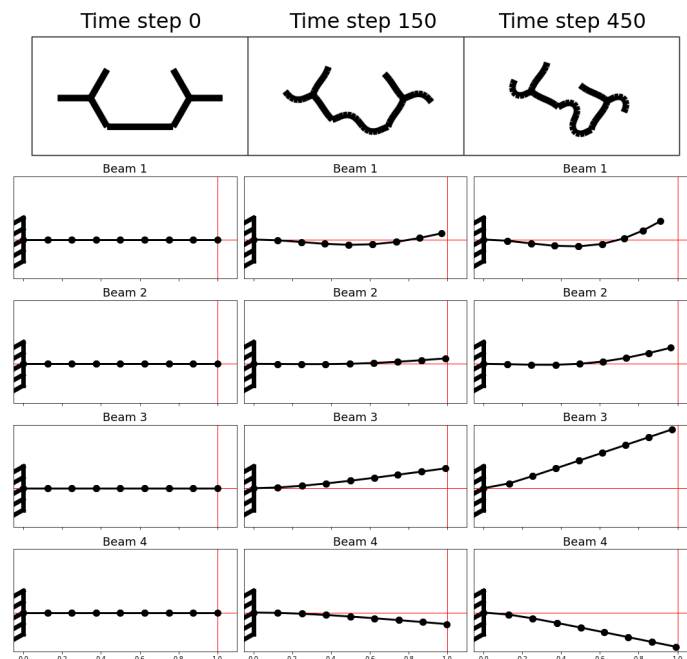


Figure 6.17: True lattice deformations and beam deformations in beam-layer for BNN-FE with 8 elements in the FE-model and a non-linear encoder, at time steps 0, 150 and 450 for a honeycomb lattice: loading case 2.

Figure 6.16 shows that, while less pronounced than the basic BNN-FE, the model still experiences difficulty capturing the buckling behaviour of the lattice. Although, the beam deformations in Figure 6.17 do show more expressive buckling behaviour. Nonetheless, given that no model selection has been performed, the revised BNN-FE does show promise of improvement and could be a good first step towards a more well-rounded surrogate model.

To summarise, while the evaluated models do not show adequate performance for practical application, the BNN-FE model, and in particular the exploratory version, shows promise to become a useful tool, given further refinement and optimization.

In this work, the possibility of replacing the microscopic model within the FE^2 framework by a surrogate model was explored, with the aim to reduce computational costs associated with modelling lattice materials. This study served as a proof of concept, with the main focus on geometric non-linearity of lattice materials, applied specifically to honeycomb and re-entrant lattices. To address common pitfalls related to black-box modelling, three physics-based artificial neural networks were developed and evaluated against a benchmark feed-forward neural network (FNN), the main objective being to increase interpretability and extrapolation capability of the model. Physics has been reintroduced through so-called beam neural networks (BNNs) that make use of beam theory: two of the physics-based models (BNN-L and BNN-NL) utilise Euler-Bernoulli beam theory for a cantilever beam, where BNN-L is limited to the linear force-displacement relation of the Euler-Bernoulli formation, and the BNN-NL introduces non-linearity by allowing changing angles of the beams with increased loading. The third (BNN-FE) employs a finite-element based approach applied on one-dimensional cantilever beams in so-called FE-boxes, solved through a static, non-linear, iterative solution procedure.

A model selection was performed and models were evaluated in both interpolation and extrapolation. No significant differences in the learning efficiency of the models were observed. Regarding accuracy in interpolation, the FNN exhibited the best performance, followed by an evenly matched BNN-FE and BNN-NL, whereas the BNN-L showed that a linear approximation of non-linear material behaviour is insufficient. A better assessment of the effects of the incorporated physics is their performance in extrapolation, showcasing their generalisation capabilities. The FNN, not supported by physics, degrades rapidly in extrapolation, while the BNNs demonstrated the ability to handle extrapolation more effectively. Although, none of the networks displayed the required performance to be used in actual FE^2 modelling: a reasonable target would be to keep prediction errors within a 10% margin, which is somewhat comparable to the macroscale simulation accuracy. This target is not achieved in this study.

To understand the lack in performance, the networks' performances were plotted in the loading space, revealing their inability to make predictions on buckled lattices. In the case of BNN-L and BNN-NL, this is due to limitations in the Euler-Bernoulli formation, however the BNN-FE is expected to accurately capture buckling, yet fails to do so. To investigate this, the interpretability of the physics-based model can be exploited, plotting the deformations of the beams inside each FE-box. It was discovered, that modelling buckling behaviour with only three elements leads to inaccurate representation of the physical response. Additionally, the absence of beam-interactions and the linear relationship between strains and displacements in the encoder weakens the compliance with physical reality.

Exploratory studies of possible improvements to the BNN-FE were carried out, to evaluate their feasibility. The first proposal was to increase the number of elements in the FE-model in the beam-layer, which had no direct effect on the buckling behaviour of the model. A second proposal was to add non-linearity to the encoder, however, this was also

ineffective. The third proposal was to combine the two previous suggestions: incorporating both a non-linear encoder and increasing the number of elements in the FE-model from 3 to 8. While each proposal on their own were unsuccessful, the combination of both showed promise of improvements to the model, but the inadequate performance of the model in buckling persists. It should be emphasised that in these exploratory studies, the model parameters of the regular BNN-FE were directly adopted, further optimization is possible and recommended.

This leads into the first recommendation for future research, namely a more extensive study the effectiveness of increasing the number of elements in the FE-model and the adoption of a non-linear encoder. Furthermore, it could be looked into to change the cantilever FE-model into a more complex variant, like a tripod (Figure 7.1). A tripod architecture could model tension and compression/buckling simultaneously, allowing for more intricate stress-strain predictions. Additionally, more complex architectures would allow beam-interconnection, linking their movements together for a realistic source of buckling initialisation. It is expected that the linking of beams through more complex FE-models will have the greatest impact on performance and should therefore be given priority in future research. Moreover, given that in this study the BNN-NL and BNN-FE models are closely matched, the BNN-NL should not be dismissed immediately. Trials could be carried out integrating analytical buckling solutions into the BNN-NL, with the aim to incorporate buckling capabilities into the model. It is clear, however, that the BNN-FE model provides greater flexibility for incorporating changes and additional features. Most notably, the BNN-FE would allow for incorporation of dynamics, given that the states of the beam-models are known at each time step, without worsening the effects of the curse of dimensionality, as is the case for RNNs. Therefore, additional research efforts into BNN-FE are justified.

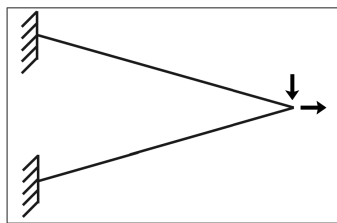


Figure 7.1: Tripod model architecture.

Should the suggestions to the current BNN-FE model bring the desired improvements, the model could gradually be exposed to more complex scenarios, such as material non-linearity, history dependency, dynamics or extending the modelling into the third dimension. However, the original purpose of the surrogate model should be kept in mind, namely it should serve a reduction in computational effort. Given that it is likely that a high number of elements is required in the BNN-FE, this negates any computational gain over the full-field analysis when a 2D honeycomb or re-entrant lattice is the subject of research. With this in mind, the BNN-FE approach might prove more useful for materials that require more intense numerical simulation, such as porous materials like foams.

To conclude, whereas the proposed models fail to demonstrate sufficient accuracy, the results show that there is room for improvement through further optimization and that there is promise for BNNs to become a useful tool to accelerate FE^2 modelling.

BIBLIOGRAPHY

- [1] Esmeralda Uribe-Lam, Cecilia D. Treviño-Quintanilla, Enrique Cuan-Urquizo, and Oscar Olvera-Silva. Use of additive manufacturing for the fabrication of cellular and lattice materials: a review. *Materials and Manufacturing Processes*, 36(3):257–280, 2021. doi: 10.1080/10426914.2020.1819544.
- [2] Christopher J. Hunt, Francescogiuseppe Morabito, Chris Grace, Yian Zhao, and Benjamin K.S. Woods. A review of composite lattice structures. *Composite Structures*, 284:115120, 2022. ISSN 0263-8223. doi: <https://doi.org/10.1016/j.compstruct.2021.115120>.
- [3] Zian Jia, Fan Liu, Xihang Jiang, and Lifeng Wang. Engineering lattice metamaterials for extreme property, programmability, and multifunctionality. *Journal of Applied Physics*, 127:150901, 04 2020. doi: 10.1063/5.0004724.
- [4] Richard Craster, Sébastien Guenneau, Muamer Kadic, and Martin Wegener. Mechanical metamaterials. *Reports on Progress in Physics*, 86(9):094501, aug 2023. doi: 10.1088/1361-6633/ace069.
- [5] Jian Xiong, Robert Mines, Ranajay Ghosh, Ashkan Vaziri, Li Ma, Arne Ohrndorf, Hans-Jürgen Christ, and Linzhi Wu. Advanced micro-lattice materials. *Advanced Engineering Materials*, 17(9):1253–1264, 2015. doi: <https://doi.org/10.1002/adem.201400471>.
- [6] Jens Bauer, Almut Schroer, Ruth Schwaiger, and Oliver Kraft. Approaching theoretical strength in glassy carbon nanolattices. *Nature Materials*, 15:438–443, 02 2016. doi: 10.1038/NMAT4561.
- [7] Ahmad Alshaer and Daniel Harland. An investigation of the strength and stiffness of weight-saving sandwich beams with cfrp face sheets and seven 3d printed cores. *Composite Structures*, 257:113391, 02 2021. doi: 10.1016/j.compstruct.2020.113391.
- [8] Krishna Kumar Saxena, Raj Das, and Emilio P. Calius. Three decades of auxetics research - materials with negative poisson's ratio: A review. *Advanced Engineering Materials*, 18(11):1847–1870, 2016. doi: <https://doi.org/10.1002/adem.201600053>.
- [9] Anirvan DasGupta. Deformation mechanics of generalized missing rib chiral lattice structures. *Composite Structures*, 344:118333, 2024. ISSN 0263-8223. doi: <https://doi.org/10.1016/j.compstruct.2024.118333>.
- [10] Joseph N. Grima and Ruben Gatt. Perforated sheets exhibiting negative poisson's ratios. *Advanced Engineering Materials*, 12(6):460–464, 2010. doi: <https://doi.org/10.1002/adem.201000005>.
- [11] Kadir Bilisik, Nesrin Sahbaz, Nedim Bilisik, Nesrin Sahbaz Karaduman, and N Bilisik. *Fiber Architectures for Composite Applications*. Springer, 01 2015. ISBN 978-981-10-0232-8. doi: 10.1007/978-981-10-0234-2_3.
- [12] Xiaobo Wang, Lei Zhang, Bo Song, Zhi Zhang, Jinliang Zhang, Junxiang Fan, Shuaishuai Wei, Quanquan Han, and Yusheng Shi. Tunable mechanical performance of additively manufactured plate lattice metamaterials with half-open-cell

- topology. *Composite Structures*, 300:116172, 2022. ISSN 0263-8223. doi: <https://doi.org/10.1016/j.compstruct.2022.116172>.
- [13] Andreas Öchsner. *Euler–Bernoulli Beam Theory*, pages 7–66. Springer International Publishing, Cham, 2021. ISBN 978-3-030-76035-9. doi: 10.1007/978-3-030-76035-9_2.
- [14] O. A. Bauchau and J. I. Craig. *Euler-Bernoulli beam theory*, pages 173–221. Springer Netherlands, Dordrecht, 2009. ISBN 978-90-481-2516-6. doi: 10.1007/978-90-481-2516-6_5.
- [15] Reza Hedayati, Naeim Ghavidelnia, Mojtaba Sadighi, and Mahdi Bodaghi. Improving the accuracy of analytical relationships for mechanical properties of permeable metamaterials. *Applied Sciences*, 11:1332, 02 2021. doi: 10.3390/app11031332.
- [16] Valentin Fogang. Euler-bernoulli beam theory: First-order analysis, second-order analysis, stability, and vibration analysis using the finite difference method. 03 2021. doi: 10.20944/preprints202102.0559.v2.
- [17] R. C. Hibbeler. *Mechanics of Materials*. Pearson Prentice Hall, 8 edition, 2010. ISBN 978-0-13-602230-5.
- [18] Gärtner Til, Fernández Mauricio, and Weeger Oliver. Nonlinear multiscale simulation of elastic beam lattices with anisotropic homogenized constitutive models based on artificial neural networks. *Computational Mechanics*, 68(5):1111–1130, 11 2021. doi: 10.1007/s00466-021-02061-x.
- [19] Filippo Masi and Ioannis Stefanou. Multiscale modeling of inelastic materials with thermodynamics-based artificial neural networks (TANN). *Computer Methods in Applied Mechanics and Engineering*, 398:115190, aug 2022. doi: 10.1016/j.cma.2022.115190.
- [20] G. Falsone and D. Settineri. An euler–bernoulli-like finite element method for timoshenko beams. *Mechanics Research Communications*, 38(1):12–16, 2011. ISSN 0093-6413. doi: <https://doi.org/10.1016/j.mechrescom.2010.10.009>.
- [21] Lovely Sabat and Chinmay Kumar Kundu. History of finite element method: A review. In *Recent Developments in Sustainable Infrastructure*, pages 395–404. Springer Singapore, 2021. ISBN 978-981-15-4576-4. doi: 10.1007/978-981-15-4577-1_32.
- [22] K. J. Bathe. *Finite Element Procedures*. Prentice Hall, 1996. ISBN 0-13-301458-4.
- [23] Robert D. Cook, David S. Malkus, and Michael E. Plesha. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, 3rd edition, 1989. ISBN 0471503193.
- [24] Boyuan Yang. Comparisons of implicit and explicit time integration methods in finite element analysis for linear elastic material and quasi-brittle material in dynamic problems, Oct 2019. URL <https://repository.tudelft.nl/islandora/object/uuid:3bf6605b-c108-4de8-a5fa-6305f7321ff4?collection=education>.
- [25] A.F. Bower. Advanced mechanics of solids. Brown University lecture notes, 2018. URL <https://www.brown.edu/Departments/Engineering/Courses/En1750/Notes/notes.html>.
- [26] Oriol Colomé, Iuri Rocha, and Frans van der Meer. *Finite Elements in Civil Engineering and Geosciences*. TU Delft faculty of Civil Engineering and Geosciences, 2023.

- [27] Tahseen Alwattar and Ahsan Mian. Developing an equivalent solid material model for bcc lattice cell structures involving vertical and horizontal struts. *Journal of Composites Science*, 4:74, 06 2020. doi: 10.3390/jcs4020074.
- [28] Karthikayen Raju, Tong-Earn Tay, and Vincent Beng Tan. A review of the fe2 method for composites. *Multiscale and Multidisciplinary Modeling, Experiments and Design*, 4(1):1–24, Jan 2021. doi: 10.1007/s41939-020-00087-x.
- [29] E. Tikarrouchine, G. Chatzigeorgiou, F. Praud, B. Piotrowski, Y. Chemisky, and F. Meraghni. Three-dimensional fe2 method for the simulation of non-linear, rate-dependent response of composite structures. *Composite Structures*, 193:165–179, 2018. ISSN 0263-8223. doi: <https://doi.org/10.1016/j.compstruct.2018.03.072>.
- [30] Frédéric Feyel and Jean-Louis Chaboche. Fe2 multiscale approach for modelling the elastoviscoplastic behaviour of long fibre sic/ti composite materials. *Computer Methods in Applied Mechanics and Engineering*, 183(3):309–330, 2000. ISSN 0045-7825. doi: [https://doi.org/10.1016/S0045-7825\(99\)00224-8](https://doi.org/10.1016/S0045-7825(99)00224-8).
- [31] Xin Liu, Su Tian, Fei Tao, and Wenbin Yu. A review of artificial neural networks in the constitutive modeling of composite materials. *Composites Part B: Engineering*, 224:109152, 2021. ISSN 1359-8368. doi: <https://doi.org/10.1016/j.compositesb.2021.109152>.
- [32] von Bartheld C. S., Bahney J., and Herculano-Houzel S. The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. *The Journal of comparative neurology*, 524(18):3865–3895, 2016. doi: <https://doi.org/10.1002/cne.24040>.
- [33] Christopher M. BISHOP. *Pattern recognition and machine learning*. Springer, 2006. ISBN 0-387-31073-8.
- [34] Christopher M. Bishop and Hugh Bishop. *Deep learning: Foundations and concepts (2024): Foundations And Concepts*. Springer Cham, 1 edition, 2024. URL <https://link.springer.com/book/10.1007/978-3-031-45468-4>.
- [35] Kevin P. Murphy. *Probabilistic machine learning: An introduction*. The MIT Press, 2022. ISBN 9780262046824.
- [36] Grant Sanderson. Backpropagation calculus, Nov 2017. URL <https://www.3blue1brown.com/lessons/backpropagation-calculus>.
- [37] Michael A. Nielsen. *Neural networks and deep learning*. Determination Press, 2015. doi: <https://doi.org/10.1007/978-3-031-29642-0>.
- [38] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2017. doi: <https://doi.org/10.48550/arXiv.1412.698>.
- [39] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. 12 2017. doi: <https://doi.org/10.48550/arXiv.1801.01078>.
- [40] Zachary Lipton. A critical review of recurrent neural networks for sequence learning. 05 2015. doi: <https://doi.org/10.48550/arXiv.1506.00019>.

- [41] Filippo Maria Bianchi, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, and Robert Jenssen. *Properties and Training in Recurrent Neural Networks*, pages 9–21. Springer International Publishing, Cham, 2017. ISBN 978-3-319-70338-1. doi: 10.1007/978-3-319-70338-1_2.
- [42] Ling Wu and Ludovic Noels. A recurrent neural network-based surrogate model for history-dependent multi-scale simulations of composite materials. CE - Commission Européenne [BE], 08 December 2021. doi: <https://doi.org/10.1016/j.cma.2020.113234>.
- [43] Dengpeng Huang, Jan Niklas Fuhg, Christian Weißenfels, and Peter Wriggers. A machine learning based plasticity model using proper orthogonal decomposition. *Computer Methods in Applied Mechanics and Engineering*, 365:113008, 2020. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2020.113008>.
- [44] F. Ghavamian and A. Simone. Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network. *Computer Methods in Applied Mechanics and Engineering*, 357:112594, 2019. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2019.112594>.
- [45] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what’s next. *Journal of Scientific Computing*, 92(3), Jul 2022. doi: 10.1007/s10915-022-01939-z.
- [46] Filippo Masi, Ioannis Stefanou, Paolo Vannucci, and Victor Maffi-Berthier. Material modeling via thermodynamics-based artificial neural networks. pages 308–329, 2021. doi: 10.1007/978-3-030-77957-3-16.
- [47] Filippo Masi, Ioannis Stefanou, Paolo Vannucci, and Victor Maffi-Berthier. Thermodynamics-based artificial neural networks for constitutive modeling. *Journal of the Mechanics and Physics of Solids*, 147:104277, 2021. ISSN 0022-5096. doi: <https://doi.org/10.1016/j.jmps.2020.104277>.
- [48] Zeliang Liu, C.T. Wu, and M. Koishi. A deep material network for multiscale topology learning and accelerated nonlinear modeling of heterogeneous materials. *Computer Methods in Applied Mechanics and Engineering*, 345:1138–1168, 2019. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2018.09.020>.
- [49] Li Zheng, Dennis Kochmann, and Siddhant Kumar. Hypercan: Hypernetwork-driven deep parameterized constitutive models for metamaterials. 08 2024. doi: 10.48550/arXiv.2408.06017.
- [50] M.A. Maia, I.B.C.M. Rocha, P. Kerfriden, and F.P. van der Meer. Physically recurrent neural networks for path-dependent heterogeneous materials: Embedding constitutive models in a data-driven surrogate. *Computer Methods in Applied Mechanics and Engineering*, 407:115934, 2023. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2023.115934>.
- [51] PA Kelly. Mechanics lecture notes: An introduction to solid mechanics., 2024. URL <http://homepages.engineering.auckland.ac.nz/~pkel015/SolidMechanicsBooks/index.html>.

A

APPENDIX: FIXED VERSUS VARIABLE BEAM ANGLES IN BEAM-LAYER

To investigate the effect of the initial angles of the beams in the beam-layer on the performance of the networks, networks with fixed initial beam angles of 0 degrees have been compared to networks with variable initial beam angles. In this case, for a network with one beam, it is set horizontally, at 0 degrees. With two beams, one is set at 0 and the other at 45 degrees. Any network with more than two beams has these beams spaced evenly between 0 and 90 degrees.

The comparison of different initial angles is only performed for BNN-L and BNN-NL models and not for the BNN-FE model, as it is expected that the initial beam angles should have no effect on the network's performance.

HYPERPARAMETERS

First, the hyperparameters of networks with varying initial beam angles must be determined. Hyperparameters for networks with fixed beam angles in the beam-layer are shown in Section 6.1. The number of curves used per dataset to characterise the hyperparameters is shown in Table A.1.

Table A.1: Number of curves per dataset for hyperparameter characterisation of networks with varying initial beam angles in the beam-layer. For both honeycomb and re-entrant lattices.

	Training	Validation
BNN-L	8000	2000
BNN-NL	8000	2000

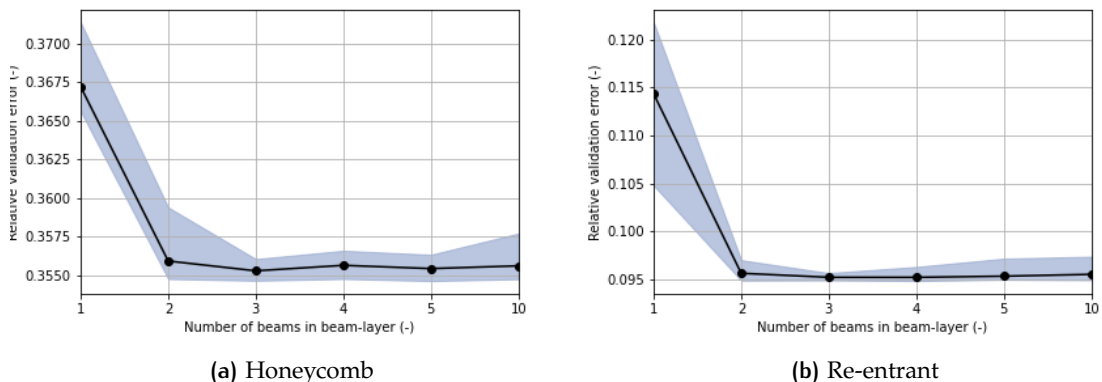


Figure A.1: BNN-L hyperparameter characterisation. Relative validation error per number of beams in the beam-layer. For a honeycomb lattice (a) and a re-entrant lattice (b).

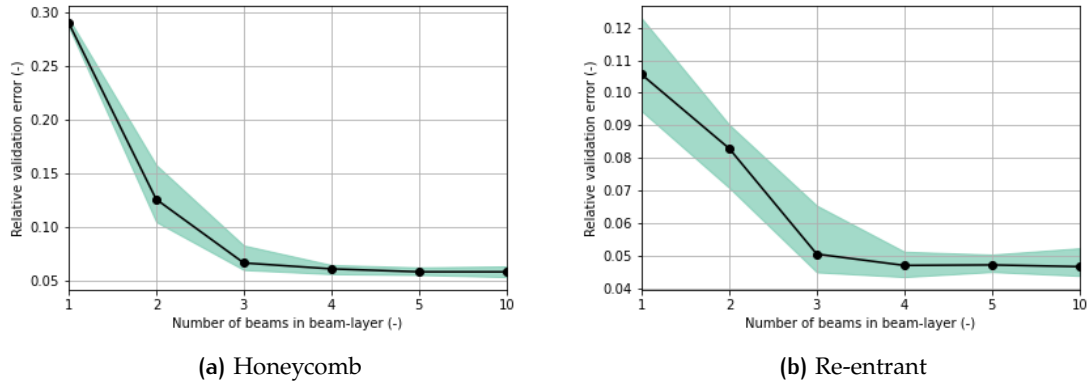


Figure A.2: BNN-NL hyperparameter characterisation. Relative validation error per number of beams in the beam-layer. For a honeycomb lattice (a) and a re-entrant lattice (b).

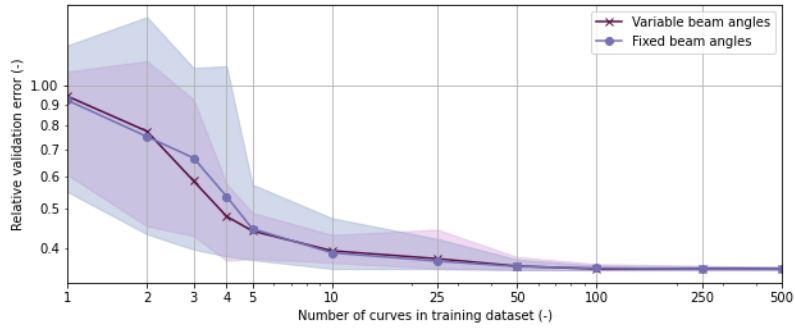
Using the results from Figures A.1 and A.2, choices can be made for the hyperparameters, these can be found in Table A.2.

Table A.2: Hyperparameter choices for BNN-L and BNN-NL with variable beam angles in beam-layer, per lattice architecture.

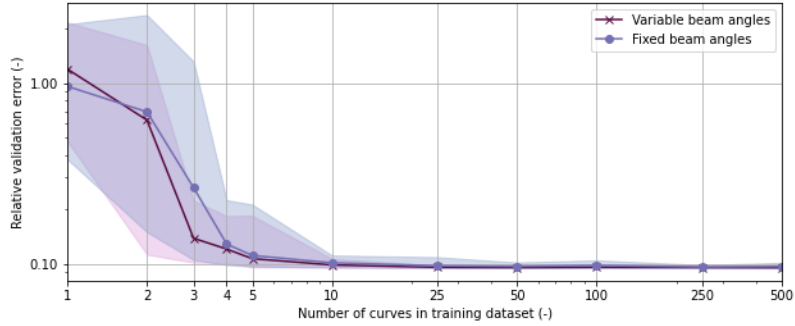
Hyperparameters		Honeycomb choice	Re-entrant choice
BNN-L	Number of beams in beam-layer	2	2
BNN-NL	Number of beams in beam-layer	3	3

LEARNING CURVES

Configuring the networks with the chosen hyperparameters as shown in Table A.2, learning curves can be created and both networks can be compared to each other directly, as the validation dataset is identical for both. Just like the hyperparameter characterisation, the validation dataset contains 2000 curves. This process is shown in Figures A.3 and A.4. It can be seen that networks behave near identically, regardless of initial beam angles. This means that the initial beam angles in the beam-layer have no effect on the performance of the network.

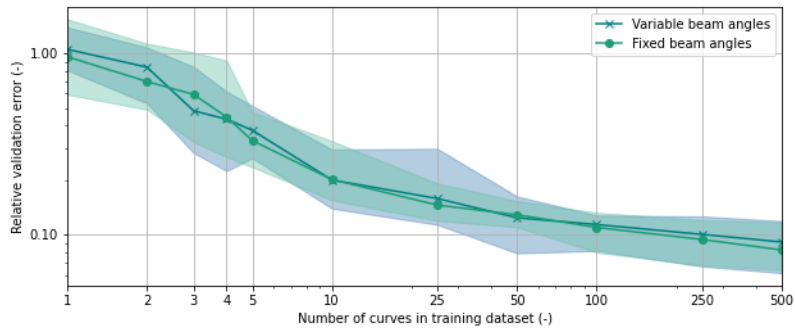


(a) Honeycomb

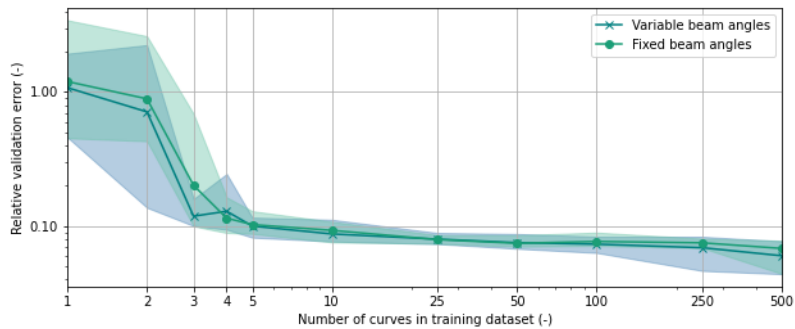


(b) Re-entrant

Figure A.3: Learning curve for fixed and variable initial beam angles in beam-layer for BNN-L network. For a honeycomb lattice (a) and a re-entrant lattice (b).



(a) Honeycomb



(b) Re-entrant

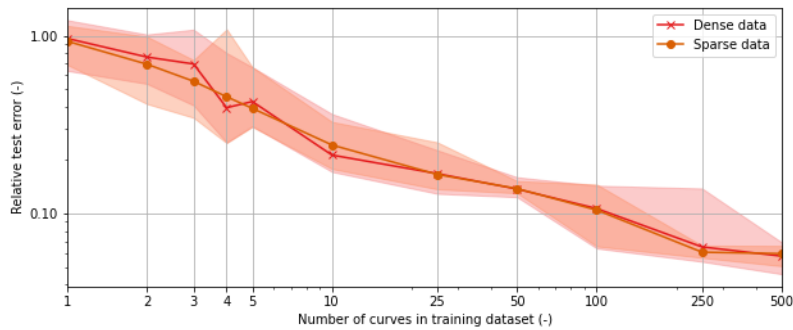
Figure A.4: Learning curve for fixed and variable initial beam angles in beam-layer for BNN-NL network. For honeycomb lattice (a) and re-entrant lattice (b).

B

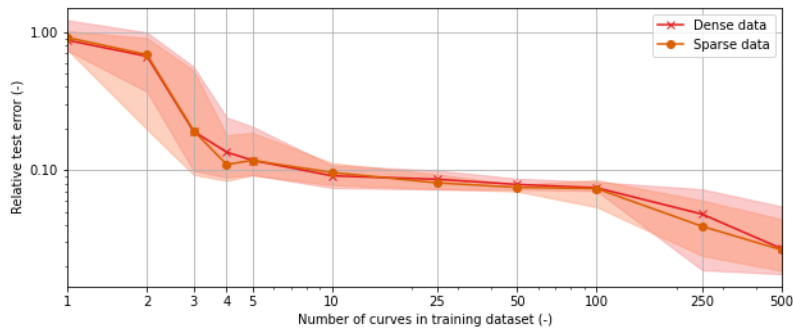
APPENDIX: SPARSE VERSUS DENSE DATASETS

In this section, networks using sparse and dense data are compared, this is only done for the FNN, BNN-L and BNN-NL networks, as the BNN-FE is only analysed using sparse data. In this work, sparse datasets are defined as datasets where every 15th data point are using along the curve. Therefore, a curve containing 150 data points, will only contain 11 data points in its sparse state. The network hyperparameters for the sparse-data networks are adopted from the dense-data equivalents, shown in Section 6.1. Then, for both the learning curves are created using the relative validation error, however these cannot be directly compared due to the composition of the validation dataset, where one dataset is sparse and contains 500 curves and the other is dense and contains 2000 curves. To compare them fairly, a test dataset containing dense data is used. In this work, this test dataset is composed of 500 curves. The networks are analysed at their point of convergence, results are shown in Figures B.1, B.2 and B.3.

From the results, it can be concluded that there is not difference in performance when using sparse or dense datasets.

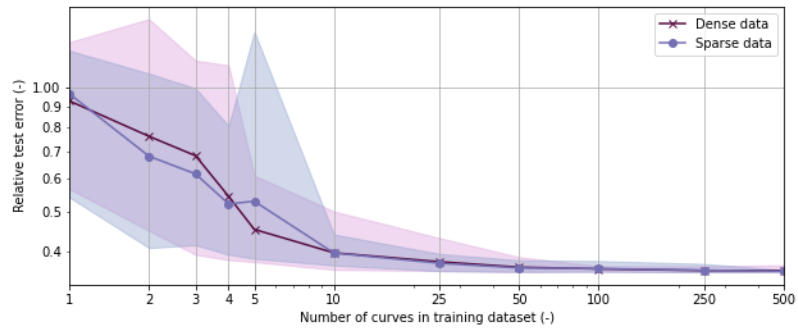


(a) Honeycomb

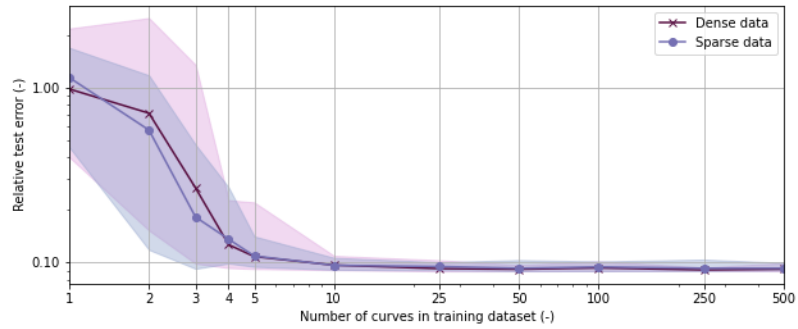


(b) Re-entrant

Figure B.1: Network assessment using sparse and dense datasets: FNN.

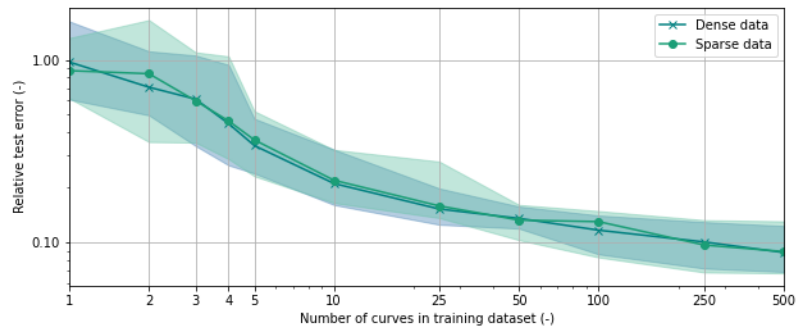


(a) Honeycomb

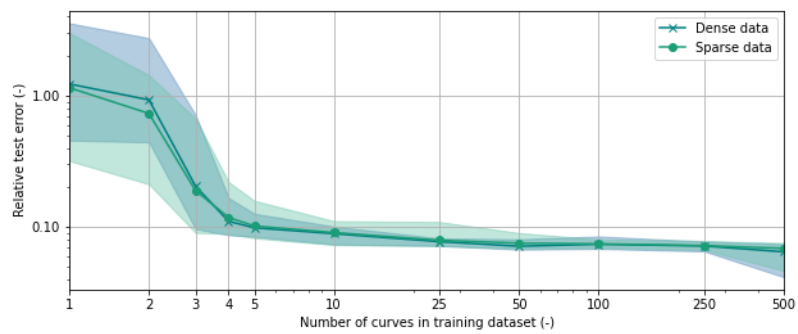


(b) Re-entrant

Figure B.2: Network assessment using sparse and dense datasets: BNN-L.



(a) Honeycomb



(b) Re-entrant

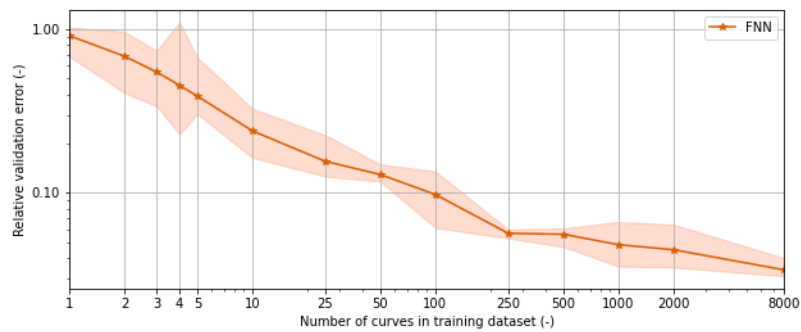
Figure B.3: Network assessment using sparse and dense datasets: BNN-NL.

C

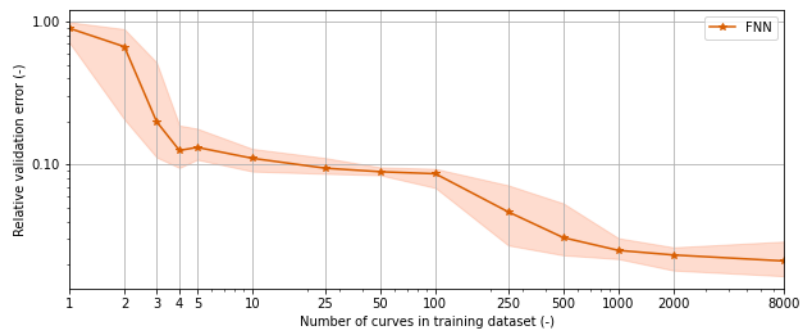
APPENDIX: SELECT, COMPLETE LEARNING CURVES

FEED-FORWARD NEURAL NETWORK (FNN)

To investigate the convergence of a FNN using sparse data, the entire learning curve is required. This is shown in Figure C.1, which is an extended version of the graph in Figure 6.5. It can be seen that the network is reasonably converged when the training dataset contains 8000 curves for the honeycomb lattice and 2000 curves for the re-entrant lattice.



(a) Honeycomb



(b) Re-entrant

Figure C.1: Extended FNN learning curve. For a honeycomb lattice (a) and a re-entrant lattice (b).

NON-LINEAR BEAM NEURAL NETWORK (BNN-NL)

Similarly to the convergence of the FNN, the full convergence of the BNN-NL is looked into. In this case, the convergence for the honeycomb lattice. The result is shown in Figure C.2 and it can be determined that reasonable convergence is reached when 2000 curves make up the training dataset.

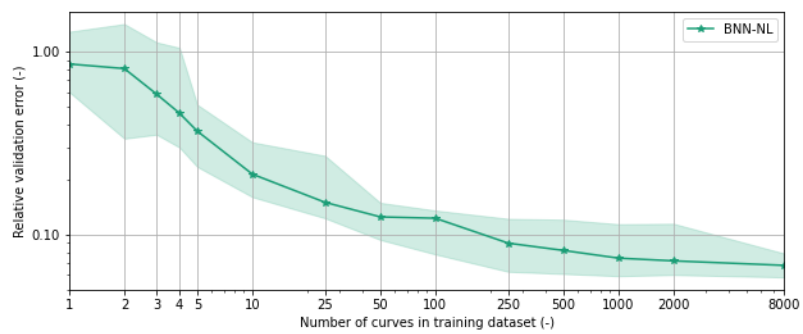


Figure C.2: Extended BNN-NL learning curve. For a honeycomb lattice.

D

APPENDIX: EXTRAPOLATION FOR A RE-ENTRANT LATTICE

Section 6.3 discussed the extrapolation performance for the honeycomb lattice architecture. In this section, results are provided for the re-entrant lattice architecture.

Similarly to what is done in Section 6.3, a test dataset containing 400 curves is created, that reach a normalised strain three times higher than in training. Relative test errors per time step can then be calculated and averaged. The result is shown in Figure D.1.

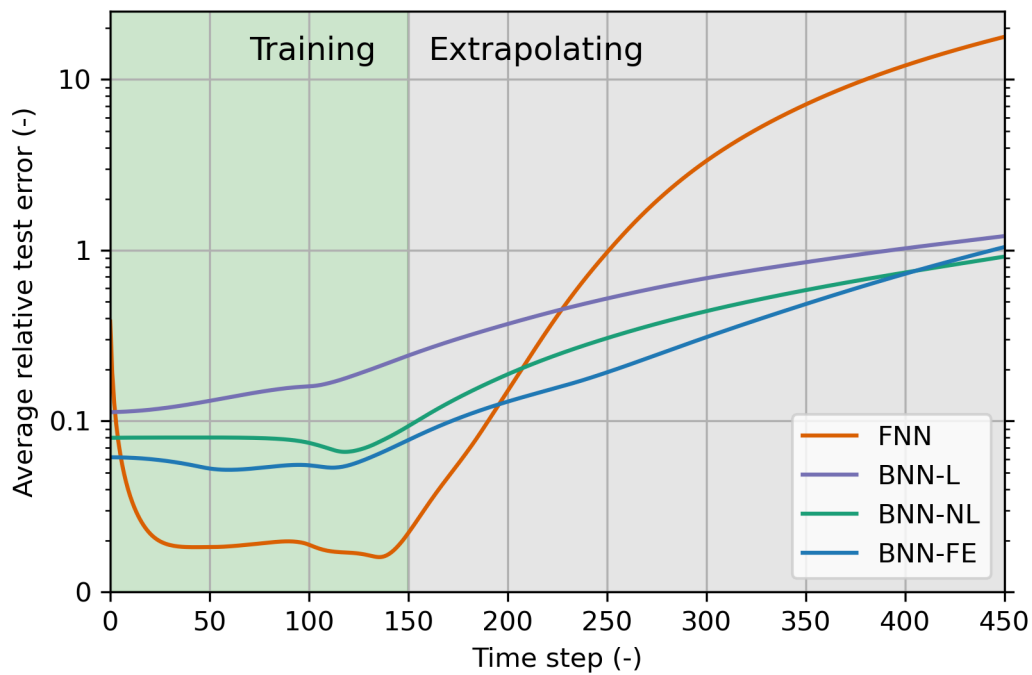


Figure D.1: Performances of networks in extrapolation: re-entrant lattice.

To gain more insight in the performances of the individual curves in the test dataset, Figure D.2 can be used. Showing that, like the honeycomb lattice in Section 6.3, there is indeed a noticeable spread in the results.

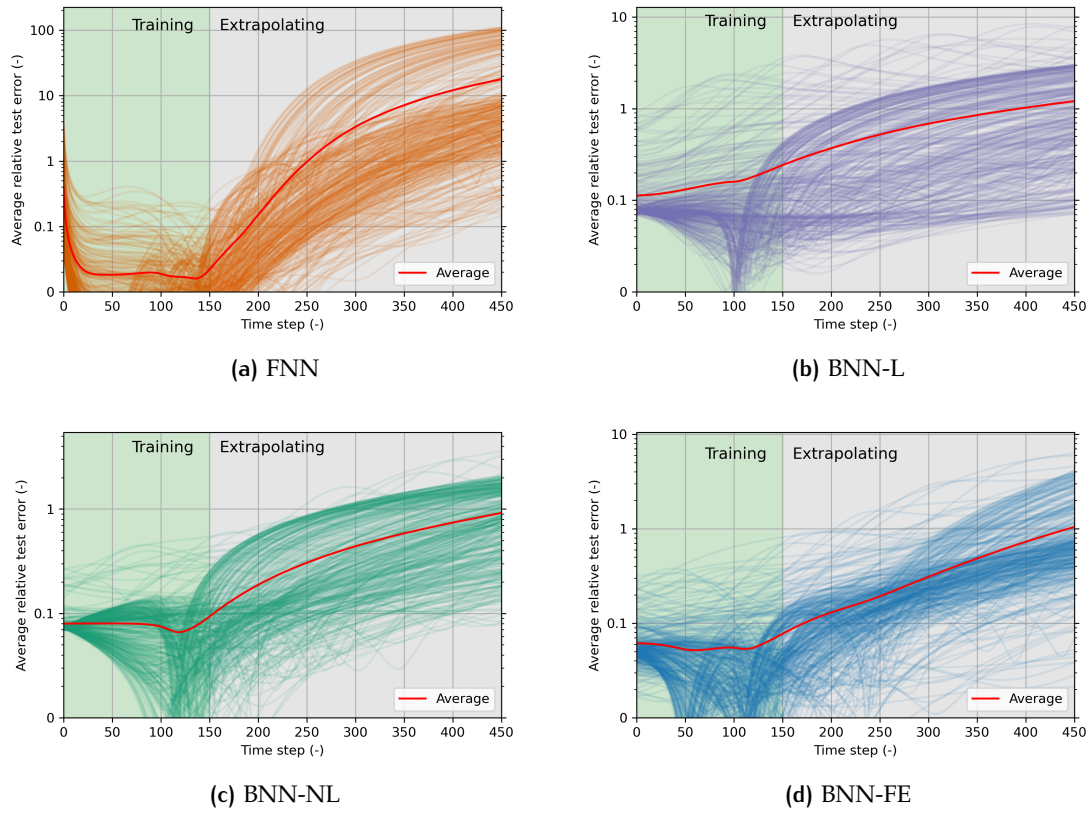


Figure D.2: Re-entrant extrapolation, all 400 test dataset curves plotted per network.

In order to gain more information on which loading cases are predicted well by the networks and which are not, plots can be generated that display the relative test error for each curve in the test dataset at a specific time step. Figure D.3 shows this for the BNN-FE at time step 200. The colours represent a discrete interval in the model's performance, from dark green (< 0.1 relative error) to dark red (> 1 relative error). Appendix E provides similar figures for more time steps and for the other networks (FNN, BNN-L and BNN-NL).

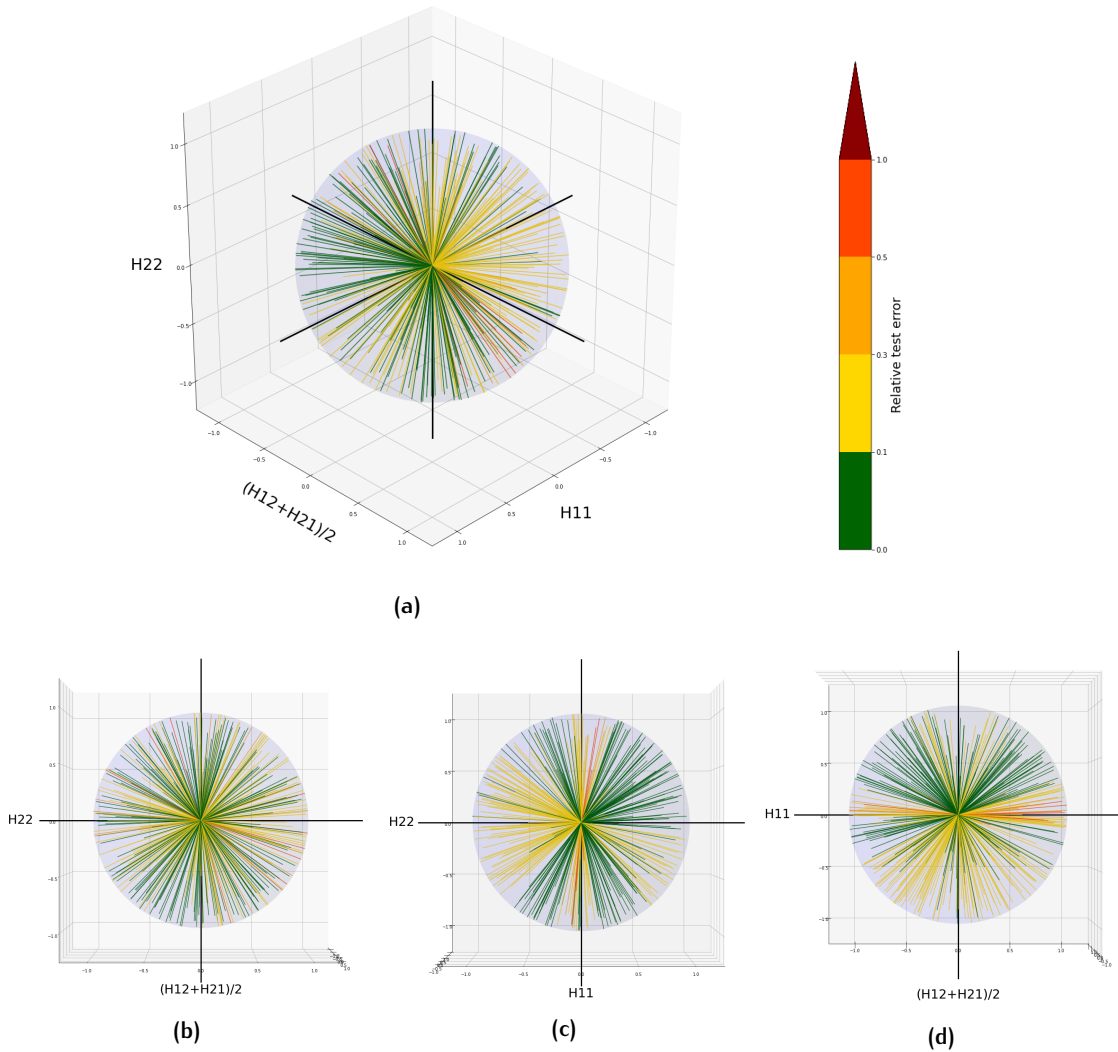


Figure D.3: Re-entrant extrapolation performance in the loading space, BNN-FE at time step 200

From Figure D.3 it can be seen that the BNN-FE struggles the most at making predictions for loading cases where there is compression in the 11 direction. This also holds for the other networks, see Appendix E. To understand why, two loading cases are investigated further, *loading case 1* is located in the region of poor fit, while *loading case 2* is located in the region of better fit. Table D.1 contains the specifics for each.

Table D.1: Loading cases: re-entrant lattice.

	H11	(H12+H21)/2	H22
Loading case 1 (poor extr.)	-0.9000	-0.3080	-0.3080
Loading case 2 (good extr.)	0.6481	0.4000	-0.6481

The network predictions for each loading case are plotted (Figures D.4 and D.6) and combined with plots showing the movements of the beams in the beam-layer in comparison with the true deformations of the lattice (Figures D.5 and D.7).

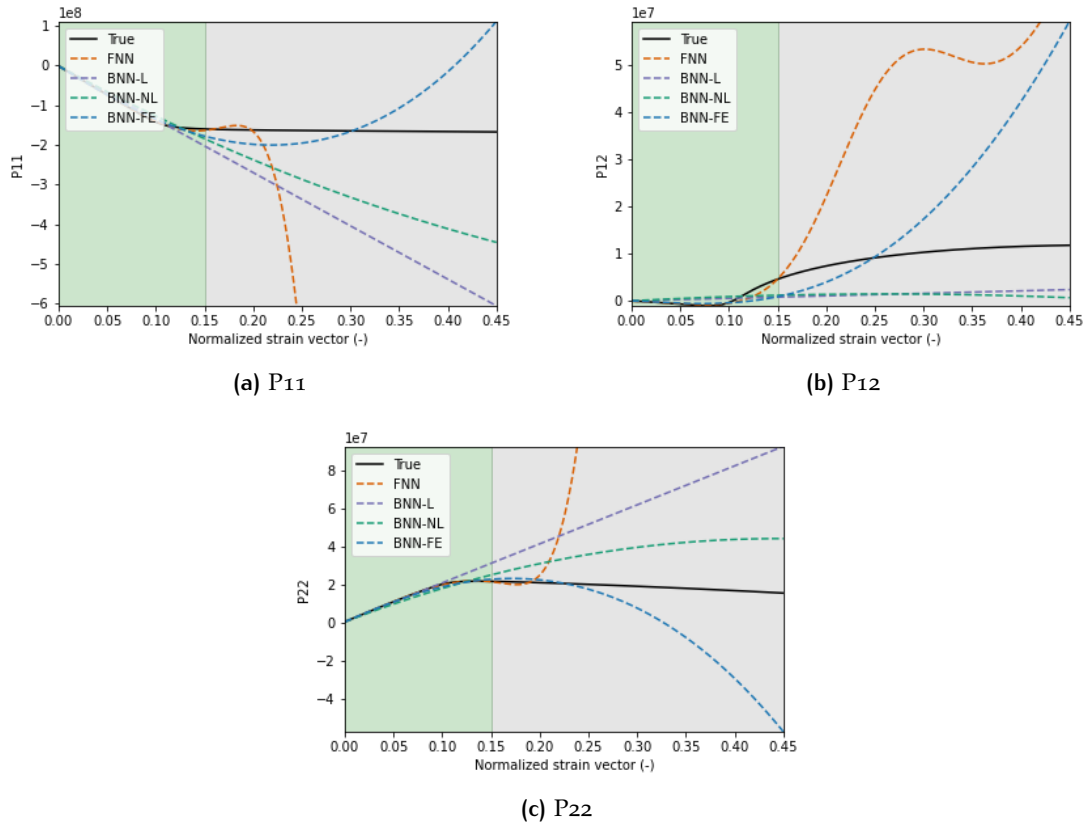


Figure D.4: Re-entrant extrapolation predictions per network: loading case 1.

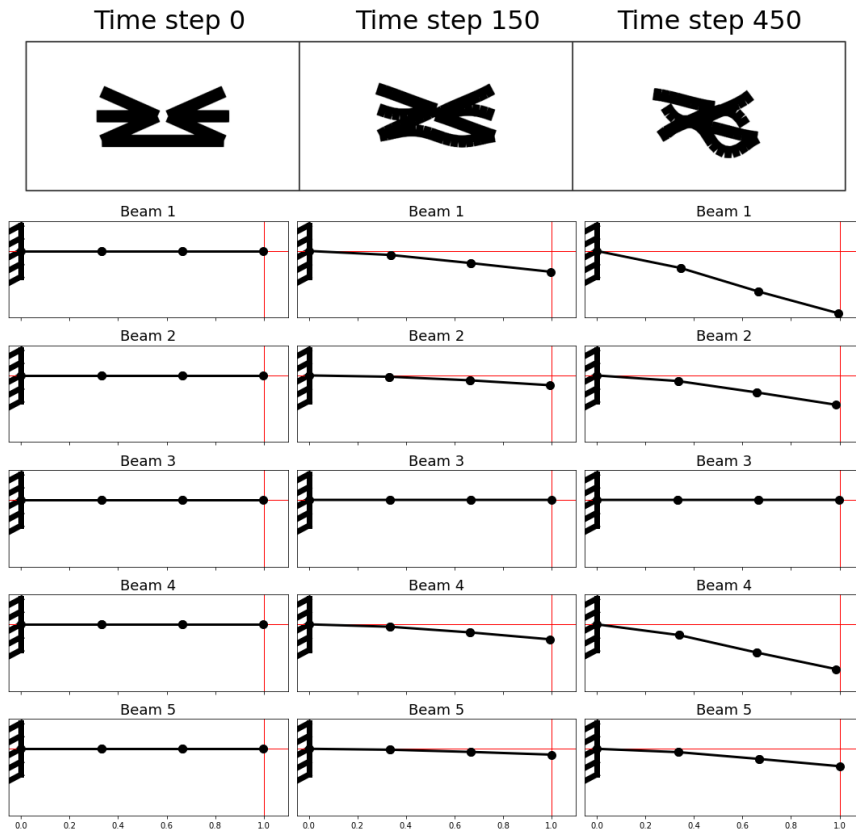


Figure D.5: True lattice deformations and beam deformations in beam-layer of BNN-FE at time steps 0, 150 and 450 for a re-entrant lattice: loading case 1.

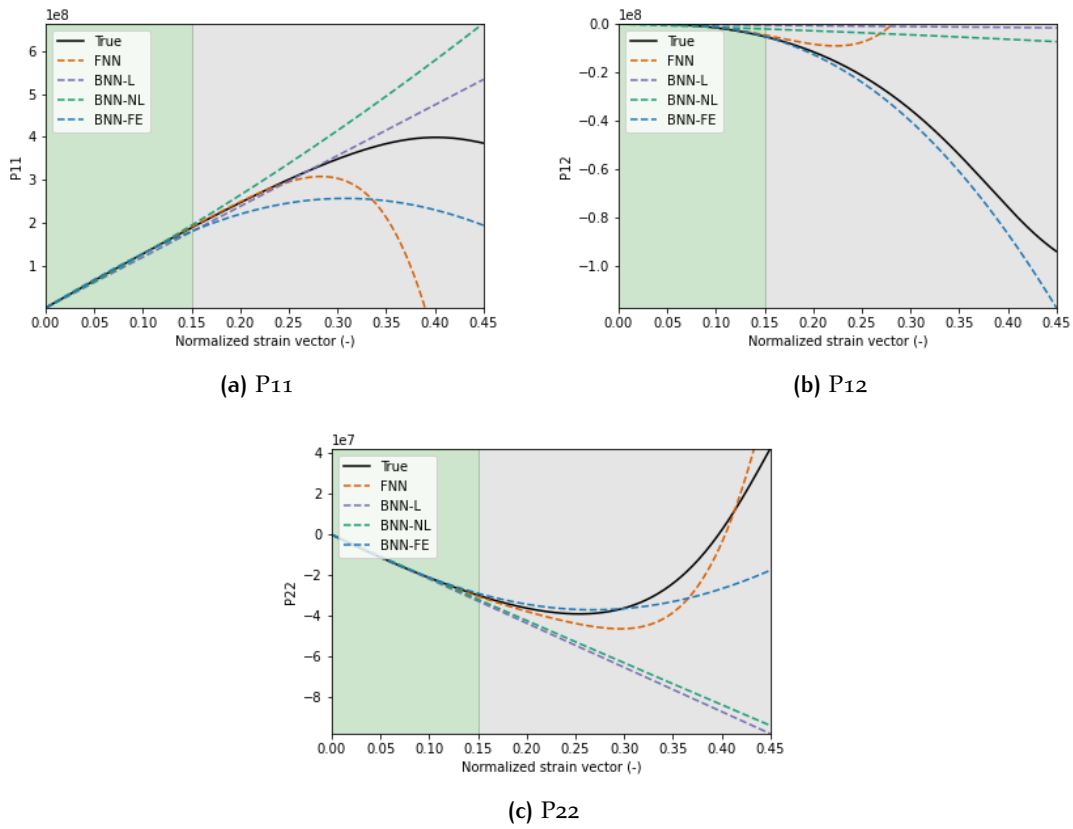


Figure D.6: Re-entrant extrapolation predictions per network: loading case 2.

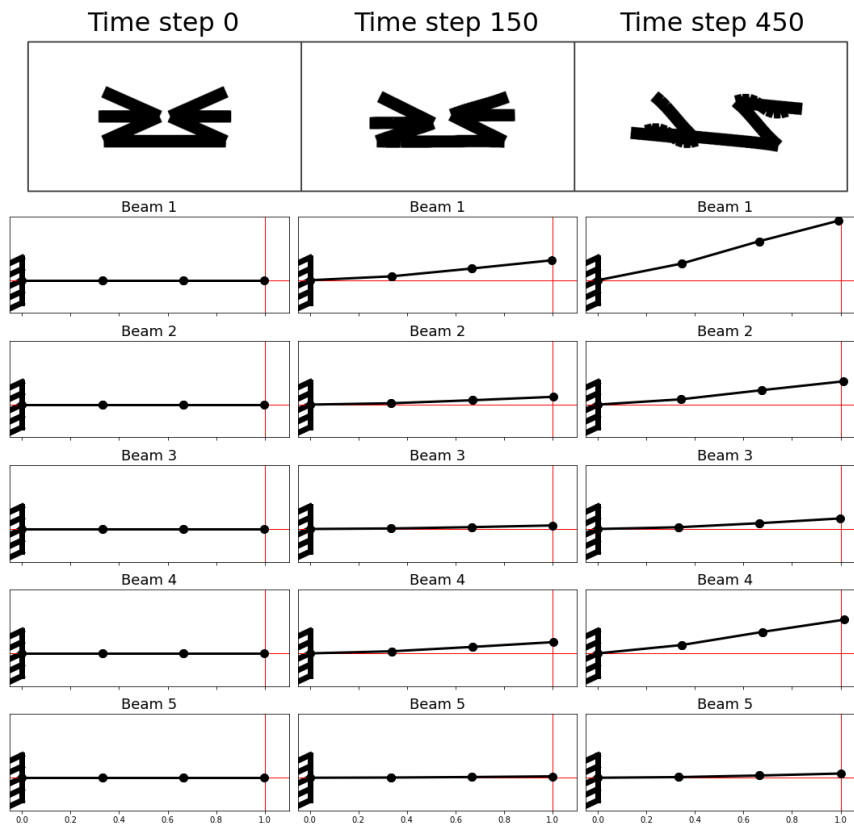


Figure D.7: True lattice deformations and beam deformations in beam-layer of BNN-FE at time steps 0, 150 and 450 for a re-entrant lattice: loading case 2.

E

APPENDIX: EXTRAPOLATION PERFORMANCE IN LOADING SPACE

In this appendix, more figures are included of each networks' performance on every curve of the test dataset at specific time steps, plotted in the loading space. The figures included show the results of both honeycomb and re-entrant lattices, at time steps 50, 100, 150 (interpolation) and time steps 200, 300, 450 (extrapolation). The colours in the plots represent a discrete interval in the model's performance, from dark green (< 0.1 relative error) to dark red (> 1 relative error).

HONEYCOMB LATTICE

Feed-forward neural network (FNN)

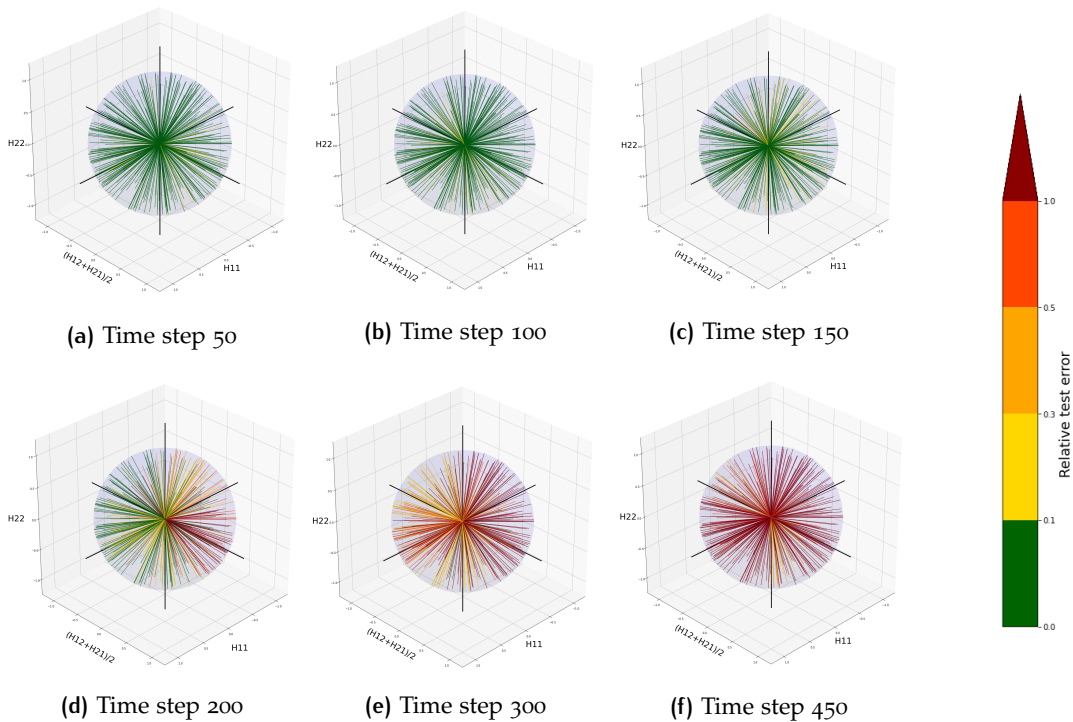


Figure E.1: Extrapolation performance in loading space: feed-forward neural network (FNN), honeycomb lattice.

Linear beam neural network (BNN-L)

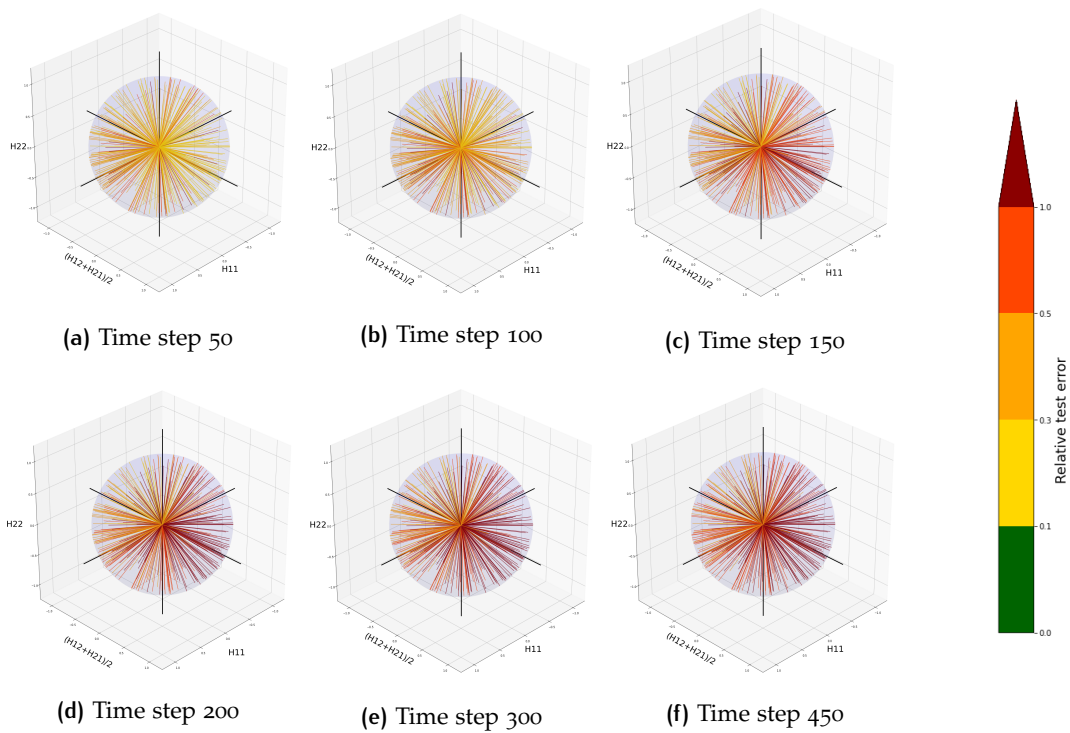


Figure E.2: Extrapolation performance in loading space: linear beam neural network (BNN-L), honeycomb lattice.

Non-linear beam neural network (BNN-NL)

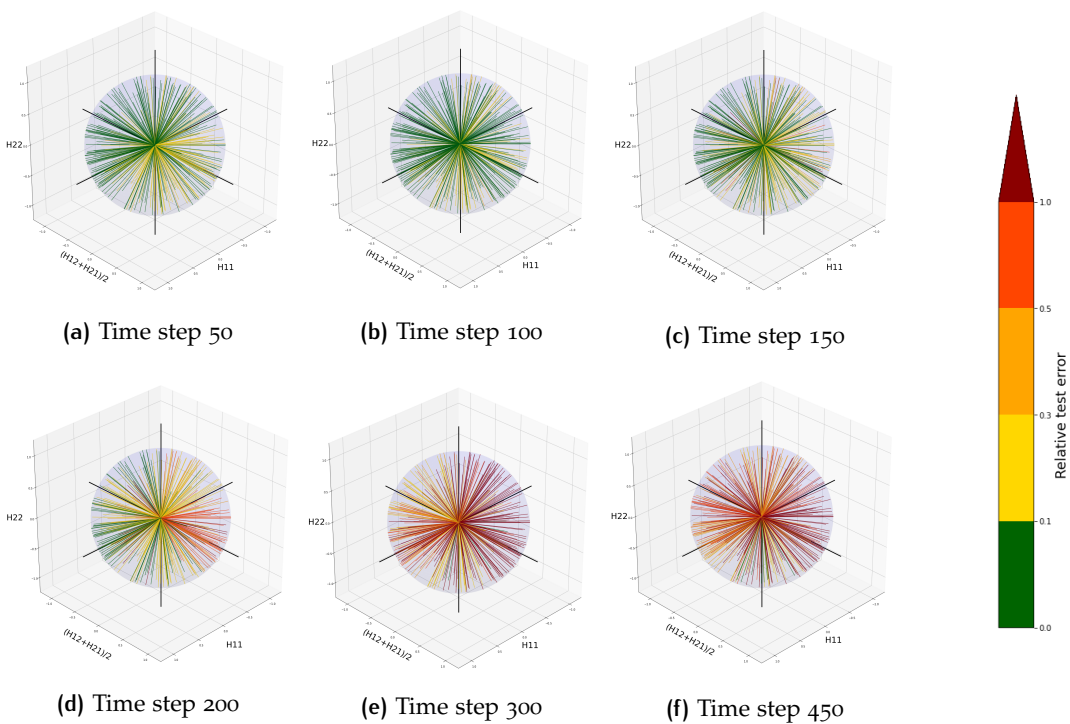


Figure E.3: Extrapolation performance in loading space: non-linear beam neural network (BNN-NL), honeycomb lattice.

Finite element beam neural network (BNN-FE)

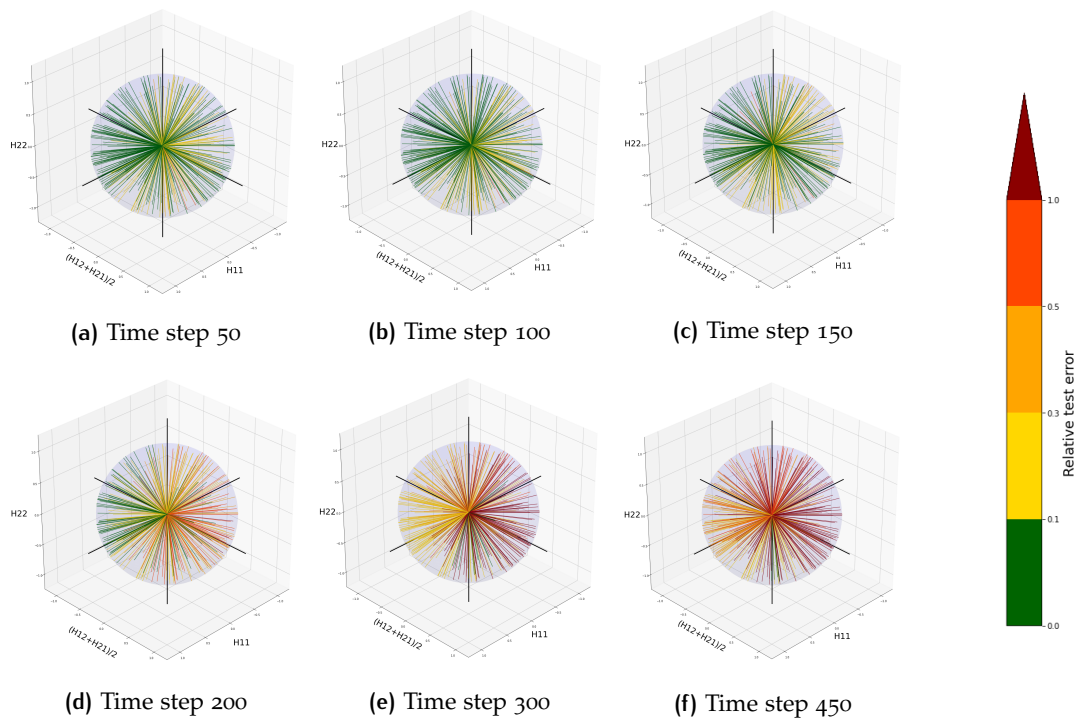


Figure E.4: Extrapolation performance in loading space: finite-element beam neural network (BNN-FE), honeycomb lattice.

RE-ENTRANT LATTICE

Feed-forward neural network (FNN)

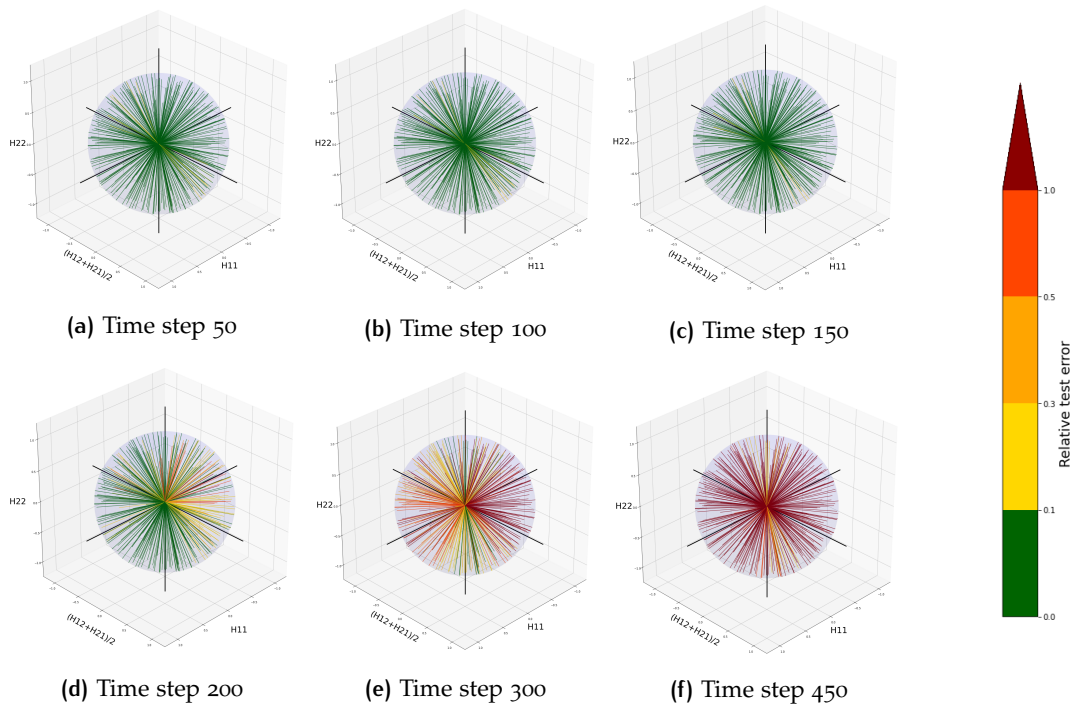


Figure E.5: Extrapolation performance in loading space: feed-forward neural network (FNN), re-entrant lattice.

Linear beam neural network (BNN-L)

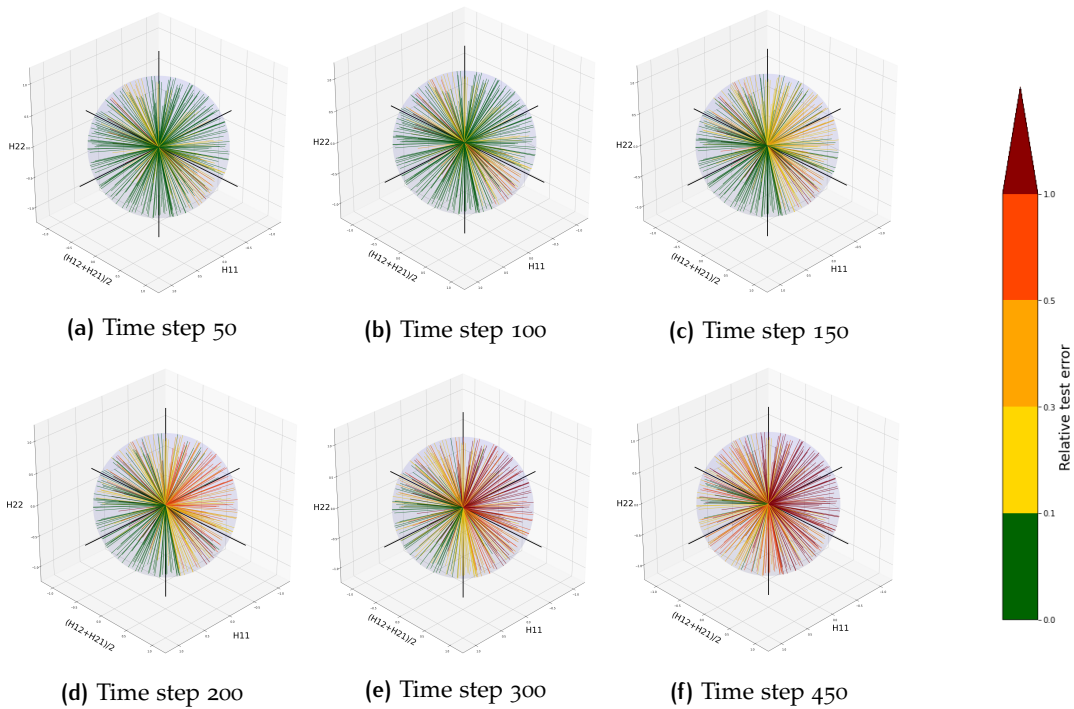


Figure E.6: Extrapolation performance in loading space: linear beam neural network (BNN-L), re-entrant lattice.

Non-linear beam neural network (BNN-NL)

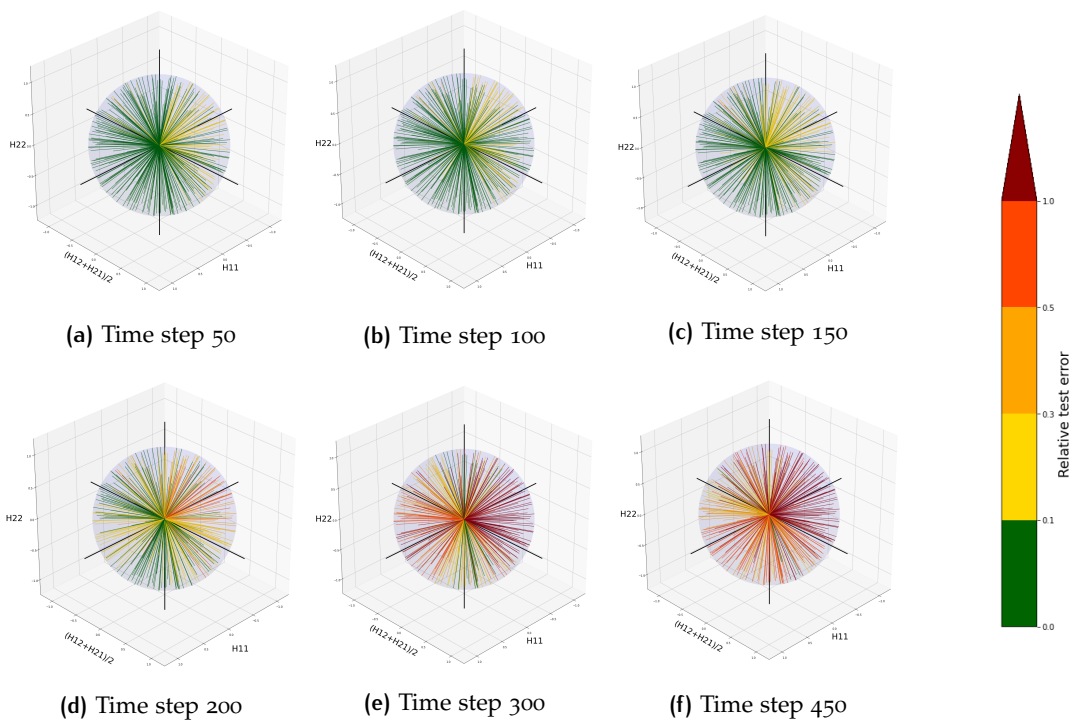


Figure E.7: Extrapolation performance in loading space: non-linear beam neural network (BNN-NL), re-entrant lattice.

Finite element beam neural network (BNN-FE)

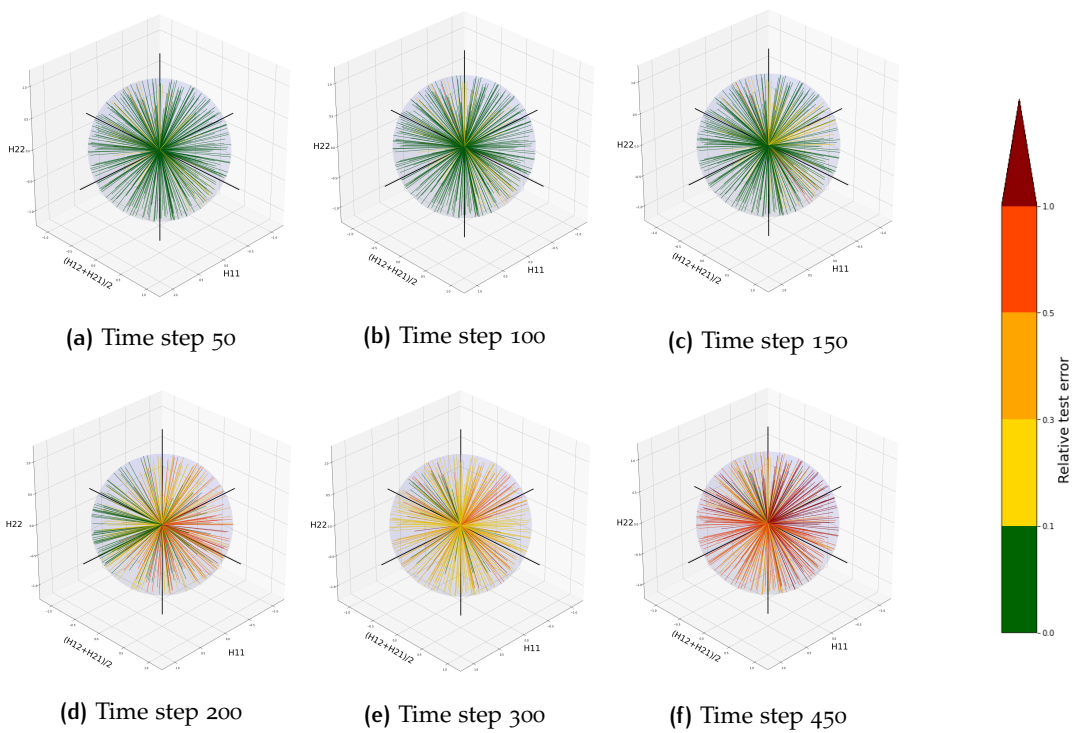


Figure E.8: Extrapolation performance in loading space: finite-element beam neural network (BNN-FE), re-entrant lattice.

F

APPENDIX: BNN-L BEAM-LAYER DEFORMATIONS

This section includes the beam deformations of the beams in the beam-layer of the BNN-L, for both the honeycomb and re-entrant lattice. Although the encoder of the BNN-L only provides endpoint deflections, deformations along the length of the beams can be deduced from the Euler-Bernoulli beam theory. Loading cases match with the ones from Section 6.3 and are repeated in Table F.1.

HONEYCOMB LATTICE

Table F.1: Loading cases: honeycomb lattice.

	H11	$(H12+H21)/2$	H22
Loading case 1 (poor extr.)	-0.7500	-0.4330	-0.5000
Loading case 2 (good extr.)	0.5000	0.7071	-0.5000

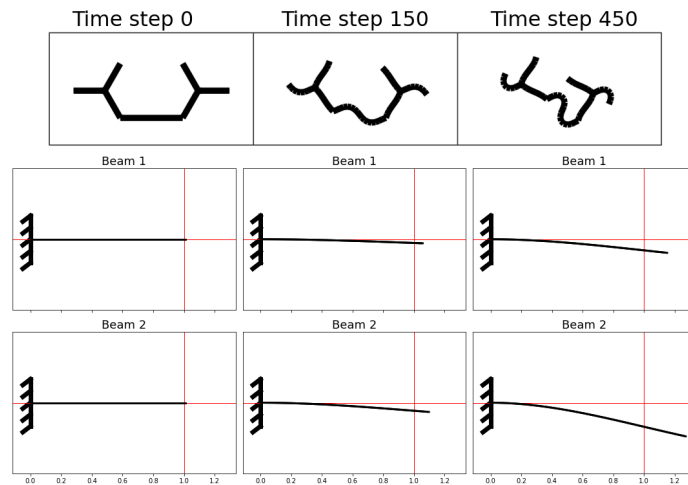


Figure F.1: True lattice deformations and beam deformations in beam-layer of BNN-L at time steps 0, 150 and 450 for a honeycomb lattice: loading case 1.

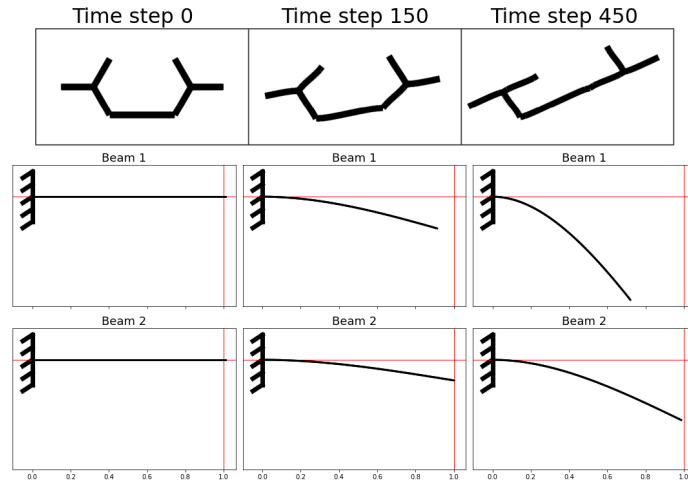


Figure F.2: True lattice deformations and beam deformations in beam-layer of BNN-L at time steps 0, 150 and 450 for a honeycomb lattice: loading case 2.

RE-ENTRANT LATTICE

Table F.2: Loading cases: re-entrant lattice.

	H_{11}	$(H_{12}+H_{21})/2$	H_{22}
Loading case 1 (poor extr.)	-0.9000	-0.3080	-0.3080
Loading case 2 (good extr.)	0.6481	0.4000	-0.6481

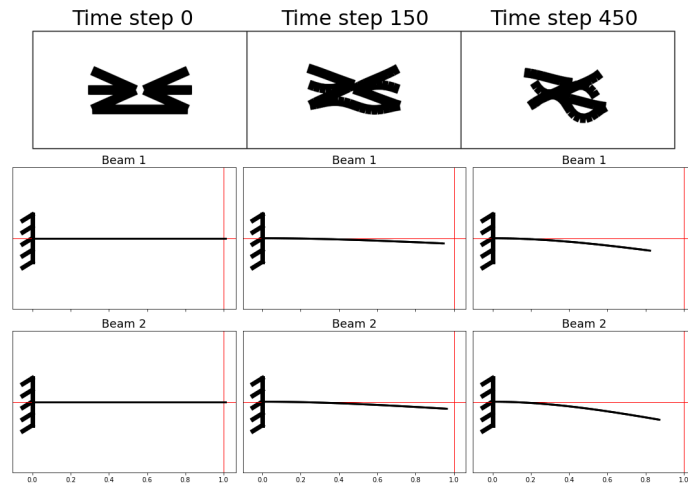


Figure F.3: True lattice deformations and beam deformations in beam-layer of BNN-L at time steps 0, 150 and 450 for a re-entrant lattice: loading case 1.

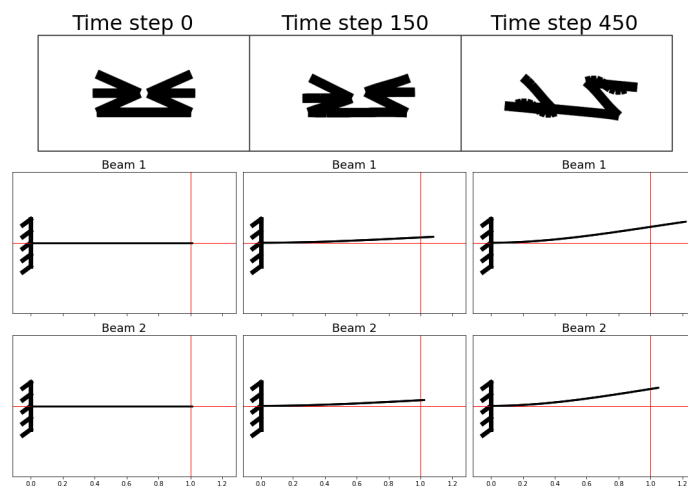


Figure F.4: True lattice deformations and beam deformations in beam-layer of BNN-L at time steps 0, 150 and 450 for a re-entrant lattice: loading case 2.

G

APPENDIX: BNN-NL BEAM-LAYER DEFORMATIONS

This section includes the beam deformations of the beams in the beam-layer of the BNN-NL, for both the honeycomb and re-entrant lattice. Although the encoder of the BNN-NL only provides endpoint deflections, deformations along the length of the beams can be deduced from the Euler-Bernoulli beam theory. Loading cases match with the ones from Section 6.3 and are repeated in Table G.1.

HONEYCOMB LATTICE

Table G.1: Loading cases: honeycomb lattice.

	H11	$(H12+H21)/2$	H22
Loading case 1 (poor extr.)	-0.7500	-0.4330	-0.5000
Loading case 2 (good extr.)	0.5000	0.7071	-0.5000

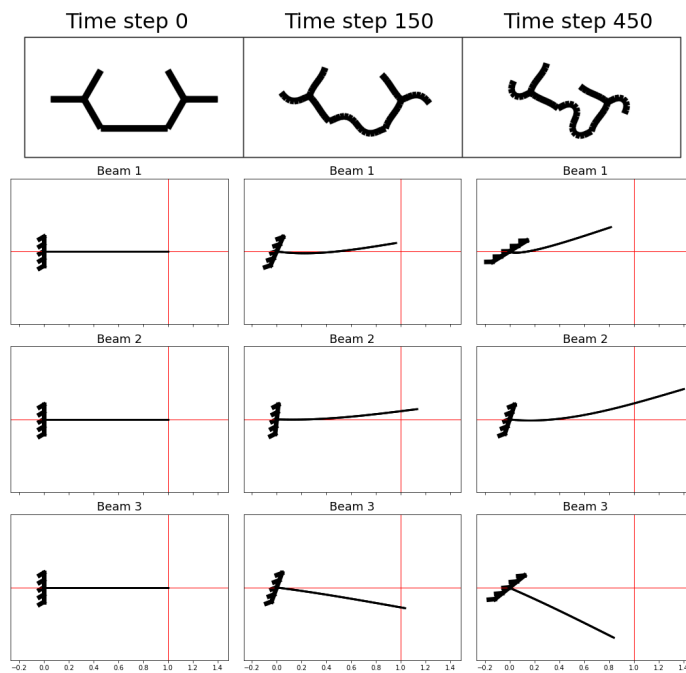


Figure G.1: True lattice deformations and beam deformations in beam-layer of BNN-NL at time steps 0, 150 and 450 for a honeycomb lattice: loading case 1.

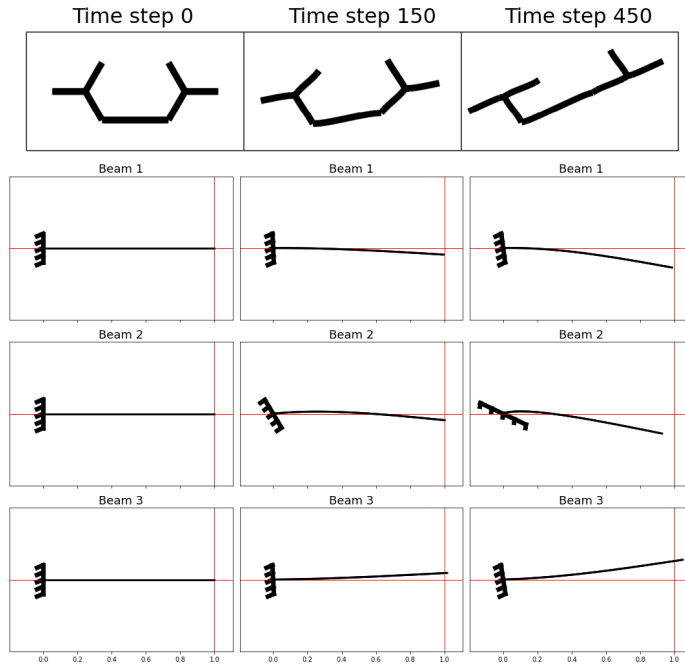


Figure G.2: True lattice deformations and beam deformations in beam-layer of BNN-NL at time steps 0, 150 and 450 for a honeycomb lattice: loading case 2.

RE-ENTRANT LATTICE

Table G.2: Loading cases: re-entrant lattice.

	H11	(H12+H21)/2	H22
Loading case 1 (poor extr.)	-0.9000	-0.3080	-0.3080
Loading case 2 (good extr.)	0.6481	0.4000	-0.6481

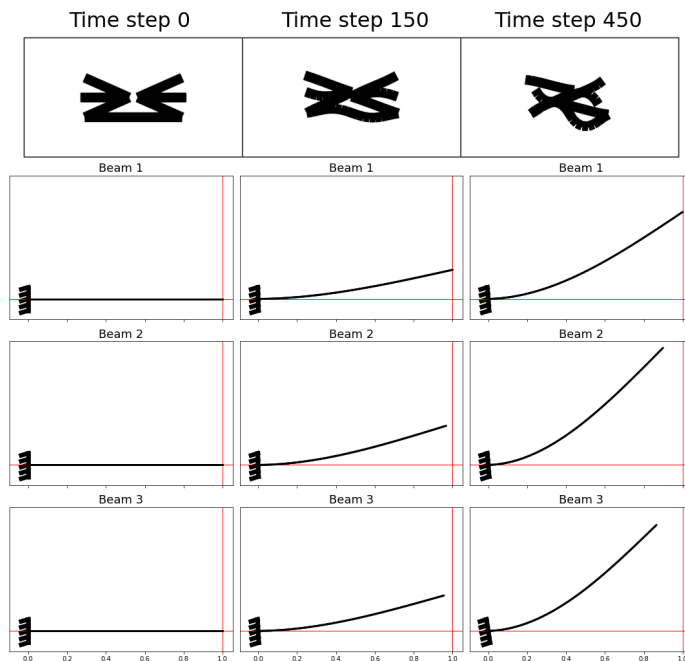


Figure G.3: True lattice deformations and beam deformations in beam-layer of BNN-NL at time steps 0, 150 and 450 for a re-entrant lattice: loading case 1.

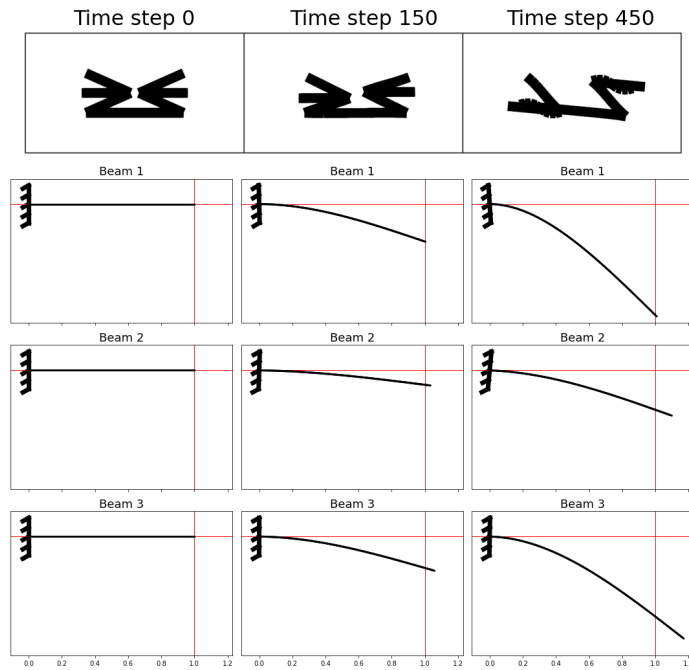


Figure G.4: True lattice deformations and beam deformations in beam-layer of BNN-NL at time steps 0, 150 and 450 for a re-entrant lattice: loading case 2.

H

APPENDIX: EXPLORATORY STUDY ON BNN-FE WITH INCREASED NUMBER OF ELEMENTS

In order to improve the buckling behaviour of the BNN-FE model, an increase of the number of elements in the FE-model is proposed. Where the original cantilever model in the BNN-FE consists of 3 elements, the following model explores the possibility to increase this to 8 elements. No other changes were made and no formal model selection was conducted; instead, the parameters of the previously developed BNN-FE were directly adopted.

The performance of the model in extrapolation is showed in Figure H.1, where the average performance of the basic, 3-elements BNN-FE model is included. For direct comparison, the same test dataset is used for both models.

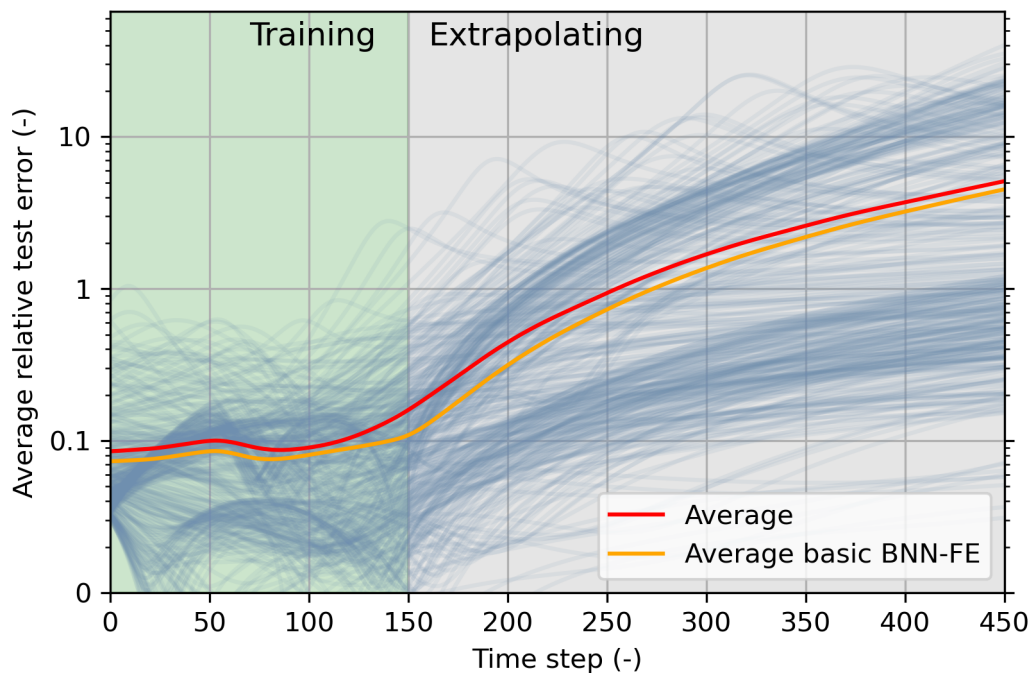


Figure H.1: Performances of BNN-FE in extrapolation, with 8 elements in FE-model: honeycomb lattice.

The model's performance on every curve of the test dataset at specific time steps, plotted in the loading space, is shown in Figure H.2. The figures included show the results at time steps 50, 100, 150 (interpolation) and time steps 200, 300, 450 (extrapolation). The colours in the plots represent a discrete interval in the model's performance, from dark green (< 0.1 relative error) to dark red (> 1 relative error). The deformations of the beams in the beam-layer for loading case 1 of Table 6.5, are shown in Figure H.3.

It is clear that no significant improvements are found compared to the original BNN-FE. The network still struggles to capture the buckling behaviour, buckling is not even observed in Figure H.3, where this is expected.

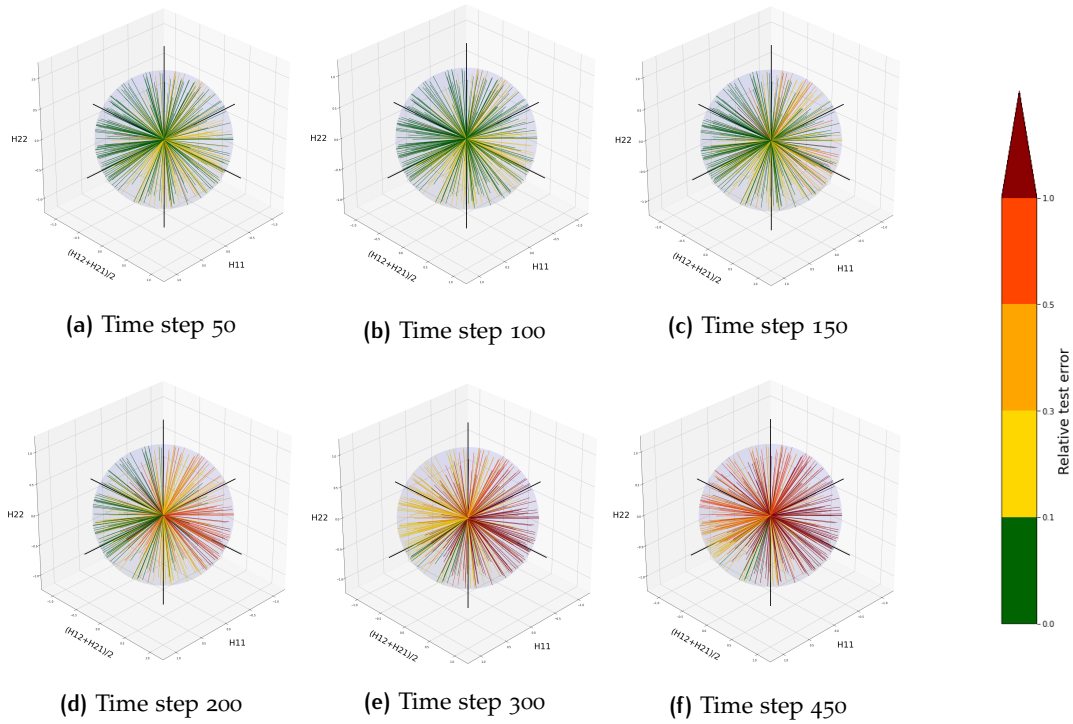


Figure H.2: Extrapolation performance in loading space: BNN-FE with 8 elements in FE-model, honeycomb lattice.

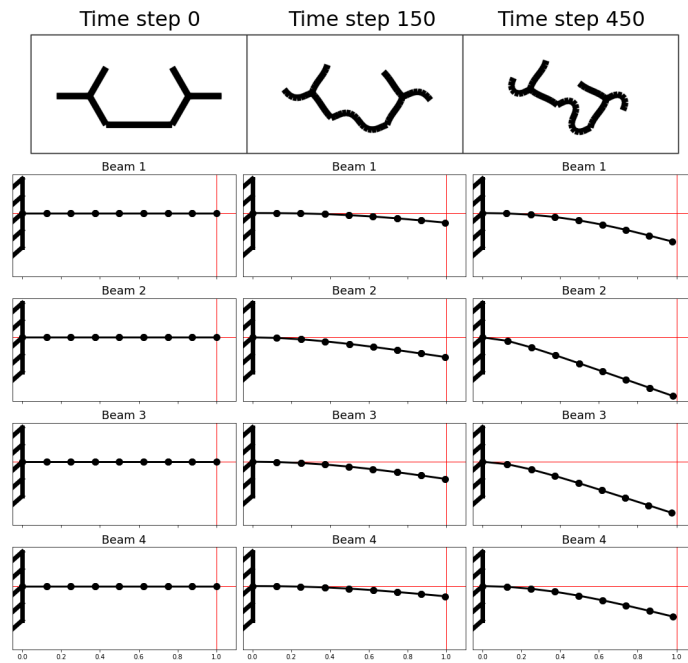


Figure H.3: True lattice deformations and beam deformations in beam-layer of BNN-NL with 8 elements in FE-model at time steps 0, 150 and 450, for a honeycomb lattice: loading case 1.

I

APPENDIX: EXPLORATORY STUDY ON BNN-FE WITH NON-LINEAR ENCODER

In order to improve the buckling behaviour of the BNN-FE model, a non-linear encoder is implemented, instead of the linear encoder used previously. The non-linear encoder is constructed by introducing an extra hidden layer after the input layer, consisting of 100 nodes and equipped with the tanh activation function. No other changes were made and no formal model selection was conducted; the parameters of the previously developed BNN-FE were directly adopted.

The performance of the model in extrapolation is showed in Figure I.1, where the average performance of the basic BNN-FE model with a linear encoder is included. For direct comparison, the same test dataset is used for both models.

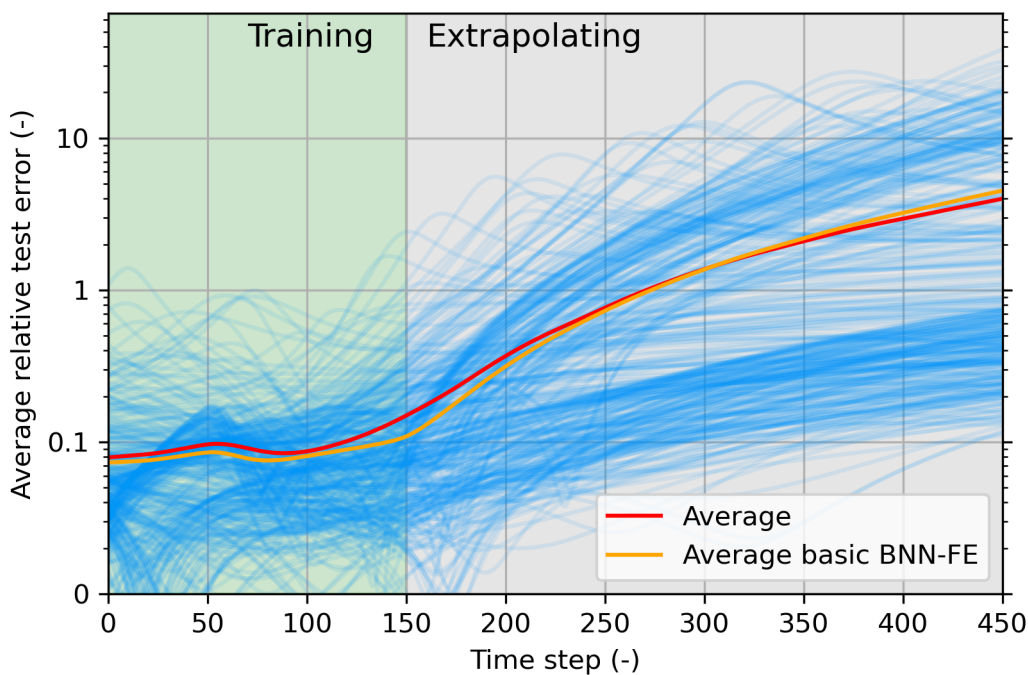


Figure I.1: Performances of BNN-FE in extrapolation, with non-linear encoder in FE-model: honeycomb lattice.

The model's performance on every curve of the test dataset at specific time steps, plotted in the loading space, is shown in Figure I.2. The figures included show the results at time steps 50, 100, 150 (interpolation) and time steps 200, 300, 450 (extrapolation). The colours in the plots represent a discrete interval in the model's performance, from dark green (< 0.1 relative error) to dark red (> 1 relative error). The deformations of the beams in the beam-layer for loading case 1 of Table 6.5, are shown in Figure I.3.

It is clear that no significant improvements are found compared to the original BNN-FE. The network still struggles to capture the buckling behaviour, buckling is not even observed in Figure H.3, where this is expected.

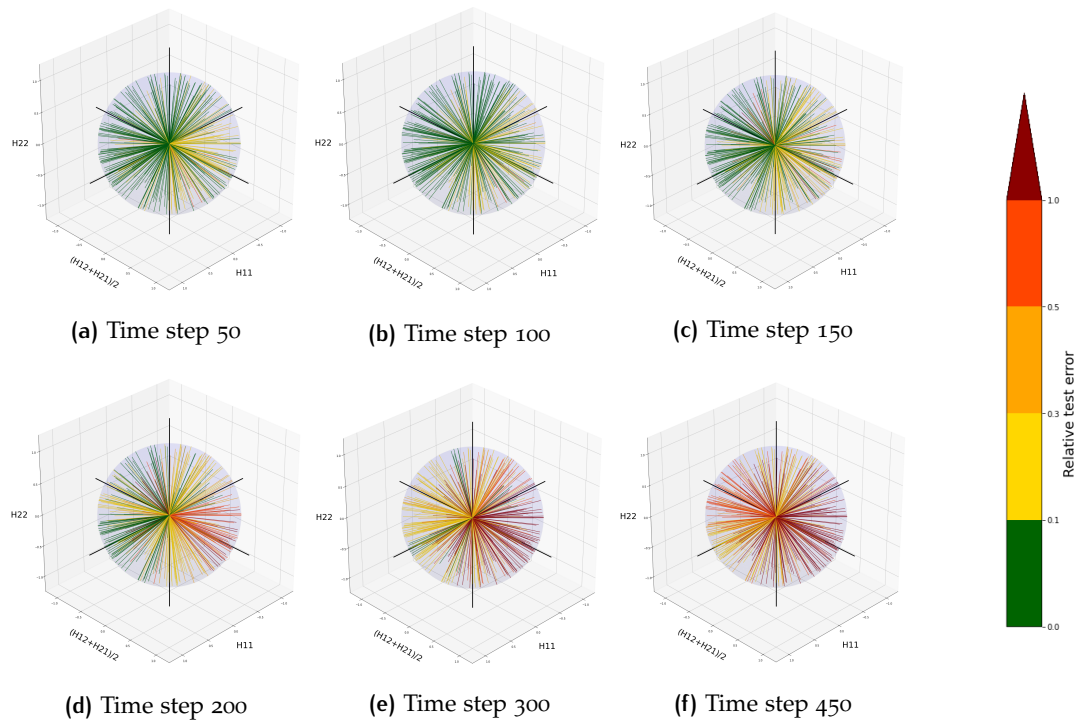


Figure I.2: Extrapolation performance in loading space: BNN-FE with non-linear encoder, honeycomb lattice.

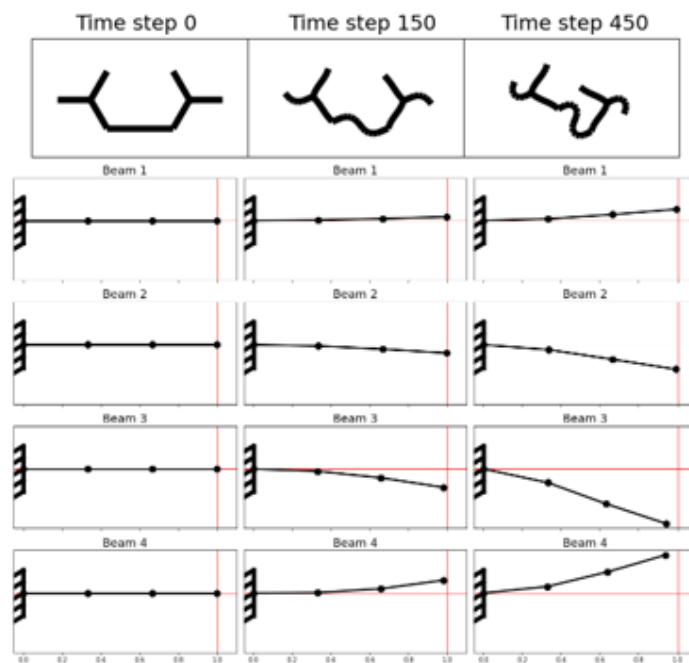


Figure I.3: True lattice deformations and beam deformations in beam-layer of BNN-NL with a non-linear encoder at time steps 0, 150 and 450, for a honeycomb lattice: loading case 1.

J

APPENDIX: EXPLORATORY NETWORK PERFORMANCE IN LOADING SPACE

Figure J.1 shows the performance of the BNN-FE network, where the FE-model contains 8 elements and the encoder is non-linear. Encoder non-linearity is obtained using an additional hidden layer after the input layer, containing 100 nodes and equipped with the tanh activation function.

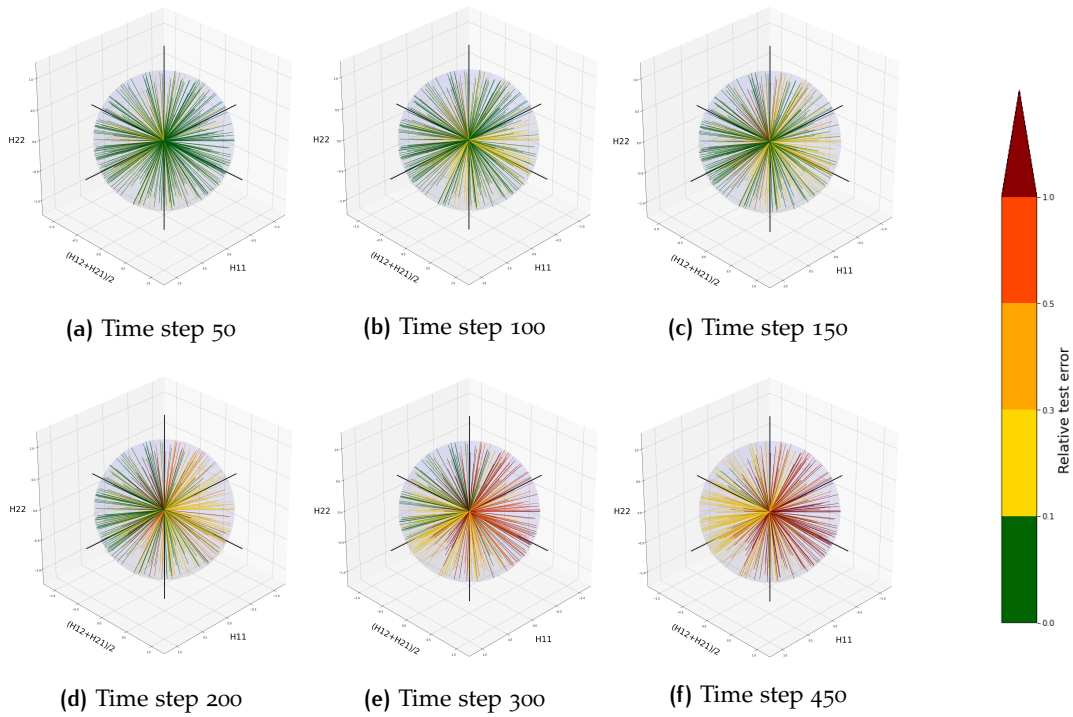


Figure J.1: Extrapolation performance in loading space: BNN-FE with 8 elements in the FE-model and a non-linear encoder, honeycomb lattice.