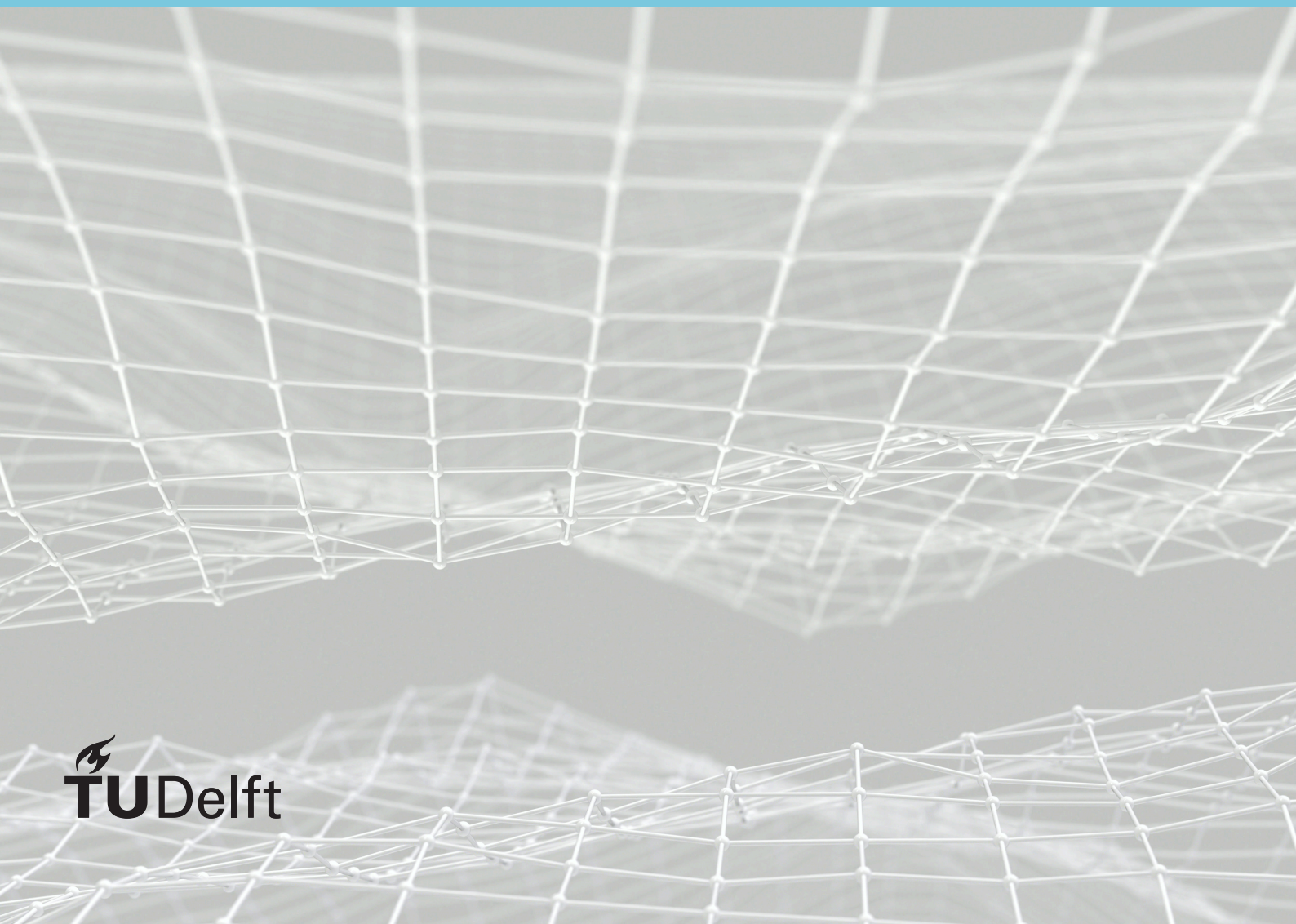
GNNs and Beam Dynamics

Investigation into the application of Graph Neural Networks to predict the dynamic behaviour of lattice beams

A. Niessen



TU Delft

GNNs and Beam Dynamics

Investigation into the application of Graph Neural Networks to predict the dynamic behaviour of lattice beams

by

A. Niessen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday December 12, 2022 at 15:30 PM.

Student number:	4692640	
Project duration:	May 1, 2022 – December 12, 2022	
Thesis committee:	Dr. ir. F. P. van der Meer,	TU Delft
	Dr. ir. R. Taormina,	TU Delft
	Ir. T. Gärtner,	TU Delft
	Ir. J. Storm,	TU Delft

Cover picture: Koi, 2022

TU Delft

Abstract

In the past decade, the application of Neural Networks (NNs) has received increasing interest due to the growth in computing power. In the field of computational mechanics, this has led to numerous publications presenting surrogate models to assist or replace conventional simulation methods. A subset of these networks, referred to as Graph Neural Networks (GNNs) impose the graph-like structure of many physical problems as a relational inductive bias. Several time-stepper implementations of these GNNs are reported to be able to simulate the dynamic behaviour of various physical objects. Within this work, it is investigated whether such GNN-based surrogate models can be applied to simulate the dynamic behaviour of lattice structures.
Upon inference of such a GNN surrogate model, the computational time required for studying lattice behaviour could be considerably reduced, thus advancing research into lattice structures as metamaterials. In addition, many large-scale structures also form a composition of beams, which could be modelled with a similar GNN.
To this end the following research question was defined: ”To what extent can GNNs be applied to simulate the dynamic behaviour of lattice structures using time-stepper methods?”. To answer this question, several GNN architectures were constructed and subsequently analyzed.

In this research, it was found that the complexity of lattice structures could not be modelled in such a way as to obtain reliable, generalisable and stable behaviour, using a time-stepper method with an architecture similar to that of Pfaff et al., 2020. It was found that due to the existence of three physically different coupled Degrees of Freedom (DOF) per node the behaviour was too complex to learn for the proposed surrogate.
At the time of writing, there is no publication presenting an effective surrogate model to simulate the dynamic behaviour of a Timoshenko beam using time-stepper methods. It is concluded that the need to capture both bending and shear behaviour using a Timoshenko beam formulation is the bottleneck for successfully modelling lattice structures.

Preface

The subject of fundamental structural mechanics has seen its last major addition in the 1920s with the formulation of the Timoshenko differential equation. For the past century, we have had a fundamental knowledge of the deformation, response and failure of most materials. However, up to this day all this fundamental physical knowledge, acquired more than a century ago is still relevant as it forms the foundation for all computational methods.
To me it is fascinating that these physical laws which describe processes like fracture, deformation and dynamics of seemingly simple continua, can only be approximated using numerical methods. Given this passion, it was at first counter-intuitive to dive into Neural Networks considering their black-box nature. Up to this day, it sometimes feels uneasy to bypass the exact relations we have got, to approximate the behaviour using a surrogate model. However, as time passed it also became apparent that designing a Neural Network is much more than throwing data at a black box. The knowledge of fundamental mechanics and numerical methods appeared to be vital to this thesis. Both for designing the presented methods and for drawing the right conclusions. In that sense, it is almost comparable to performing non-linear FEM simulations, where it seems like the computer does all the work, but for the right modelling choices and interpretations a thorough understanding of the fundamental knowledge is essential.

I would like to thank Til Gärtner and Joep Storm, for composing the topic of this thesis and for sharing their knowledge and support throughout the duration of this thesis. Furthermore, I thank Frans van der Meer as chair of the committee and Riccardo Taormina as external supervisor for their involvement, suggestions and supervision.

A. Niessen
Delft, November 2022

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

DOF	Degree Of Freedom
EER	Energy Error Ratio
FEM	Finite Element Method
GNN	Graph Neural Network
MLP	Multi-Layer Perceptron
MPL	Message-Passing Layer
MSE	Mean Squared Error
NRMSE	Normalized Root Mean Square Error
NN	Neural Network
PDE	Partial Differential Equation
RMSE	Root Mean Square Error

List of Symbols

$\mathbf{A}$	Adjacency matrix of a graph
A	Area of cross-section
$\mathbf{B}$	Derivative shape function matrix
b	Bias term
$\mathbf{D}$	Constitutive matrix (material definition)
Δt	Time increment
E	Set of edges
E	Youngs-modulus
$\mathcal{E}$	Energy in system
θ	Angle of a beam or edge with the global x-axis
G	Graph
G	Shear-modulus
$\mathbf{g}$	Vector of a single edge-embedding
γ	Shear-angle
$\mathbf{H}$	Matrix of node embeddings for the entire graph
$\mathbf{h}$	Vector of single node-embedding
$\mathbf{h}$	Vector of hidden layer
$\mathbf{I}$	The identity matrix
I	Moment of inertia (Cross-section)
$\mathbf{K}$	Global stiffness matrix
k	Extensional spring-stiffness
κ	Curvature of the line of deflection for a beam
κ_1	Effective shear-area correction factor
l	Length
$\mathbf{M}$	Global mass matrix
M	Moment
m	Point-mass
m	Number of excluded time-steps at beginning of simulation
$\mathcal{N}$	Set of neighbours
$\mathbf{N}$	Shape function matrix
ν	Poisson's ratio
ρ	Material density
σ	Nonlinear activation function
σ	Standard deviation
$\mathbf{u}$	Vector of displacements
V	Set of nodes
V	Shear force
w	The lateral displacement of the beam center-line
ϕ	Phase angle
ϕ	Angle of cross-section rotation
$\mathbf{W}$	Matrix containing learnable weights
ω	Radial frequency
$\mathbf{x}$	Vector describing location in global Cartesian space
x	Location in global x-direction
$\overline{x}$	Location in local x-direction
z	Location in global z-direction

1 Introduction

Within the field of computational mechanics, a trade-off exists between model complexity, reliability and computational cost. When analyzing the mechanical behaviour of structures, a large variety of modelling decisions have to be made. These decisions can have direct implications on the numerical accuracy and the level of physical complexity which is captured. However, enhancing the model quality generally leads to an increase in computational cost. Taking this trade-off into consideration, the reduction of computational cost enables the modelling of larger systems in combination with complex physics.

A particular application in which there is a need to model complex physical processes is the research into the behaviour of engineered lattice structures. These meta-materials are designed to exhibit material behaviour which cannot be easily found in nature. In the specific case of lattices, this is done by connecting beams in a repetitive configuration. There are several of these repetitive configurations which result in negative Poisson's ratios, such as the re-entrant profile or the rotating square profile (Francisco et al., 2021). This auxetic material behaviour has been of great interest recently because of its energy absorption capabilities.

To further research the behaviour of these lattices in events where high energy absorption is relevant, numerical simulations need to be performed. The deformation behaviour in these processes is largely determined by physical and geometrical non-linear effects which significantly increases the model complexity. While the simulation of these effects can already be performed using Finite Element Methods (FEM), the large computational cost of the numerical calculations limits the ability to perform research on these materials.

One approach which has the potential to reduce the cost of such simulations without significantly compromising model accuracy and complexity is the application of Neural Networks (NNs). This Machine Learning (ML) technique has been shown capable of simulating physical processes within the field of computational mechanics since the 1990s with a recent surge in the past 5 years (Yagawa and Oishi, 2021). In some studies, the Neural Network is only used as a surrogate for part of the model calculation, such as modelling a non-reflective boundary using NNs in order to reduce the model size (Ziemiański, 2003). Another example is the application of an NN scheme to get stable results for dynamic explicit time integration schemes with larger time-steps (Meister et al., 2020). However, there are also more recent studies which train on data obtained by FEM software and construct a surrogate model to predict the behaviour of the entire continuum (Pfaff et al., 2020; Sanchez-Gonzalez et al., 2020).

The latter mentioned studies make use of a subset of NNs called Graph Neural Networks (GNNs). These networks distinguish themselves by introducing a relational inductive bias which arises from the graph-like properties of the continuum (Battaglia et al., 2018). This inductive bias forces the network to learn the continuum behaviour based on the relevant neighbour information. Furthermore, GNNs can be applied using a permutation invariant approach and therefore have the potential to generalize to unseen graphs of different size.

The objective of this thesis is to investigate whether Graph Neural Networks can be applied to simulate the dynamic behaviour of 2-dimensional lattice structures with unseen geometries. The subsequent intended use of such a model would be to study the dynamic behaviour of auxetic lattice structures under impact loads with reduced computational cost.

In order to simulate dynamic problems using Neural Networks, two types of methods can be distinguished (Legaard et al., 2021). Firstly there are 'Direct solvers' which learn the trajectory of a physics problem with the initial conditions embedded in the weights, and can consequently approximate the state of the dynamic system at any given time. Secondly, there are 'time-stepper models', which take the dynamic state and try to learn the physical behaviour to make a next-step prediction. The advantage of the latter strategy is its potential capability to predict the behaviour of structures with unseen initial conditions. This property is essential for the application of the NN in different impact events without the need for expensive retraining. Therefore, only time-stepper models are investigated within the scope of this report.
There are various studies which implement these time-stepper methods to simulate physical behaviour with Graph Neural Networks. Firstly, Sanchez-Gonzalez et al., 2020, presented a surrogate model to simulate dynamic particle interaction of fluids and granular solids. The proposed time-stepper method was reported to be stable upon rollout with qualitatively good results. However, the physics behind particle interaction is fundamentally different from the beam dynamics which are discussed in this report. Pfaff et al., 2020, proposed a time-stepper surrogate model to simulate mesh-based dynamics. Their surrogate can simulate the quasi-static deformation of a plate modelled as a three-dimensional continuum and the dynamics of cloths in combination with contact. It is stated that the presented surrogate model is two orders of magnitude faster than traditional solvers while showing "visually plausible" behaviour after long rollouts.

Considering advances closely related to modelling lattice structures in particular, it is valuable to take a closer look into the present knowledge with respect to the simulation of Timoshenko beam behaviour. Given the scope of this report, it is vital to capture both shear and Bernoulli beam behaviour because of the dimensions of the lattice beams and the variety of wavelengths present in high-impact events.
Within current literature, a handful of studies predicting this Timoshenko behaviour using Neural Networks can be found. Firstly, there are several studies applying NNs to predict the frequencies of a Timoshenko beam with attached masses (Yõldõrõm, 2014 & Demirdag and Murat, 2009) and continuous micro-beams (Rajasekaran et al., 2022). These are not as relevant to the proposed solution method in this report, since they do not simulate the behaviour itself, but only predict some physical properties. Secondly, Papadopoulos et al., 2018 presents a method to make next-step predictions for a Timoshenko beam element. Their work introduces a surrogate NN model which predicts the forces within a deformed geometrically non-linear Carbon Nano Tube. These internal forces are in turn used in the Newton-Raphson method, which infers that they are applied in a quasi-static fashion. Aside from this, the shear behaviour is included through a modification of the bending stiffness. While this might be a valuable approximation given the constraints within the presented research, it is not an accurate method when evaluating deformations with several different wavelengths.
Regarding the modelling of meta-materials in general, Xue et al., 2022 proposed a surrogate model to simulate the dynamic behaviour of soft mechanical meta-materials. This work discretizes the meta-material into a graph with rigid crosses as nodes and springs as edges. Consequently, it proposes Gaussian Process Regression and Multi-Layer Perceptrons (MLPs) as methods to approximate the highly nonlinear potential energy within the springs. This potential energy can be substituted into the Lagrangian formulation of the discretized system, which upon numerical integration gives a new kinematic state.
While their work is similar to the proposed method in this report, because it predicts the dynamic behaviour of meta-materials using a time-stepper method, there are also notable differences. Firstly, the surrogate replaces a continuum model so there is no regard for Timoshenko beam dynamics. Secondly, the connection between nodes within the material is only discretized by one spring which decreases model complexity. Lastly, the model only updates the edges using graph information, after which it calculates the total energy. It does not use Message Passing Layers and only uses the graph structure to get the local and global energy.
The proposed method in this work has the potential to be more flexible upon inference when dealing with different geometries. Furthermore the discretization of lattice beams in multiple elements enables the model to capture behaviour with smaller wavelengths.

The presented research shows advances in modelling dynamics, complex beam behaviour and metamaterials. However, there is a lack of publications combining these complexities. The primary goal of this report is to investigate the possibility to model the dynamic behaviour of lattice structures using GNNs. Therefore the main research question is: ”To what extent can GNNs be applied to simulate the dynamic behaviour of lattice structures using time-stepper methods.” To answer this question several sub-questions will be elaborated on.

- What architectural choices are relevant when modelling dynamics using GNN time-stepper models?
- How well can a GNN time-stepper model generalise to different configurations in space?

To answer the proposed questions several GNN architectures were implemented using PyTorch Geometric (Paszke et al., 2019 & Fey and Lenssen, 2019). The GNN architectures were evaluated and the results from the most informative ones are shown in this report. The data needed to train the NNs was obtained by running implementations of a FEM scheme. For the continuum simulations Wells, 2020 can be used as a reference and for the Timoshenko simulations Eugster, 2015 can be consulted. It is important to note that the research questions will be answered based on the behaviour of the designed surrogate models. Given the heuristic approach, which is common for machine learning applications, it is not possible to give definite general qualifications.

This report is structured as follows: Firstly a brief introduction to Graph Neural Networks is given in Chapter 2. After which a surrogate model is presented which models the dynamic behaviour of a one-dimensional continuum in Chapter 3. Subsequently, Chapter 4 presents a surrogate model for modelling beam dynamics using the Timoshenko beam theory and shows the difficulties which arise. Lastly, the discussion and conclusion will elaborate on the research questions.

2

Graph Neural Networks

In order to get acquainted with Graph Neural Networks (GNNs), this chapter will introduce some key concepts which will be used in Chapter 3 to model the behaviour of a one-dimensional continuum and Chapter 4 to model the behaviour of Timoshenko-beams. The following information is by no means an extensive explanation of Neural Networks in general but rather a summary of knowledge which holds direct connection to methods which are proposed later.
The chapter will start with a basic introduction into Neural Networks, followed by a short description of Graphs and concluding with a section on Graph Neural Networks.

2.1. Neural Networks

Neural Networks (NNs) are a widely used technique in the field of machine learning. By using linear matrix multiplications in combination with nonlinear activation functions, a Neural Network can predict an outcome $\mathbf{v}$ from an input $\mathbf{q}$, where the relation between in- and output is a nonlinear function.
There are numerous more conventional examples of applications for this method. However, to be in line with the subject of this report we will consider a 2 DOF mass-spring system in series with a certain initial displacement. The desired output $\mathbf{v}$ is a two-dimensional vector consisting of the respective acceleration of each node.

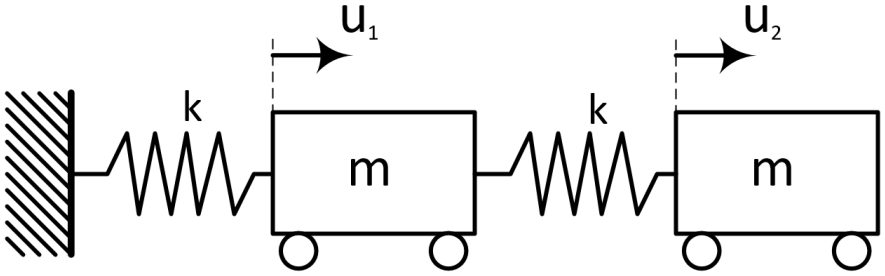


Figure 2.1: The 2-DOF mass-spring system used as a visual example.

2.1.1. Input features

In order to design a NN which can solve this problem we need to determine the relevant properties that influence it. For example, to predict the acceleration it is essential to know the current displacement field. Other inputs are the value of the masses which cause inertia and the stiffness of the springs which cause forces. These inputs are referred to as 'input features'.
In various applications of NNs it is not always clear which input features are needed to get the best prediction. However, in this report the focus is on acquiring a network that mimics processes for which close to exact solutions can already be found. From these solutions it can already be deduced with which input features a solution can be achieved. This is contrary to applications of NNs where there

is no pre-existing model. In these cases much more uncertainty exists as to the correlation between possible input features and the predicted outcome.
By gathering the proposed features a four-dimensional feature-vector $\mathbf{q} = [u_1, u_2, m, k]^T$ is constructed. Where the first pair of values indicate the displacements of the respective DOFs, the third value indicates the weight of the masses and the fourth value indicates the stiffness of the springs. Important to note here is that all data of the whole system is gathered inside one feature vector.

2.1.2. Neural Network Architecture

In order to predict the output given the defined input we assume that there is some combination of matrix multiplications and non-linear mappings which performs this task satisfactorily. Explained more mathematically, the following process is followed. Given the input $\mathbf{q} \in \mathbb{R}^n$ we construct a matrix $\mathbf{W_1} \in \mathbb{R}^{n \times d}$ and multiply to get a new 'hidden' vector $\mathbf{h_1} \in \mathbb{R}^d$. This 'hidden' vector[1] is transformed by some nonlinear mapping which is known as an 'activation function'. Subsequently, the same procedure is repeated using a second weight matrix $\mathbf{W_2} \in \mathbb{R}^{d \times d}$ to get a second hidden vector $\mathbf{h_2} \in \mathbb{R}^d$. Lastly, to get a vector which represents the desired output, another weight-matrix multiplication is needed, $\mathbf{W_3} \in \mathbb{R}^{d \times m}$, where the shape is defined by the size of the output vector $\mathbf{v} \in \mathbb{R}^m$. This example, which is displayed in Eq. 2.1 and Figure 2.3 only has 2 hidden layers, by adding more multiplications and activation functions the number of layers can be expanded, giving more sophistication to the network. Each pair of matrix-multiplication in combination with an activation function creates a new 'layer' which can be referred to as a 'hidden' layer. A composition of multiple of these layers is referred to as a Multi-Layer Perceptron (MLP).

$$\mathbf{q} \rightarrow \sigma\left(\mathbf{W_1} \cdot \mathbf{q}\right) \rightarrow \mathbf{h_1} \rightarrow \sigma\left(\mathbf{W_2} \cdot \mathbf{h_1}\right) \rightarrow \mathbf{h_2} \rightarrow \mathbf{W_3} \cdot \mathbf{h_2} \rightarrow \mathbf{v} \tag{2.1}$$

If there would only be matrix multiplications there is no possibility to output anything other than linear combinations of the input features. This renders the use of multiple layers useless and would be identical to linear regression. To prevent this from happening the activation functions σ are used. These functions take an input and perform some nonlinear transformation. A few common activation functions are shown in Figure 2.2. The ReLU-function is most used in the proposed schemes with regard to dynamic modelling.

$$ReLU\left(x\right) = max\left(0, x\right) \tag{2.2}$$

$$LeakyReLU\left(x\right) = max\left(\alpha x, x\right) \tag{2.3}$$

$$Sigmoid\left(x\right) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

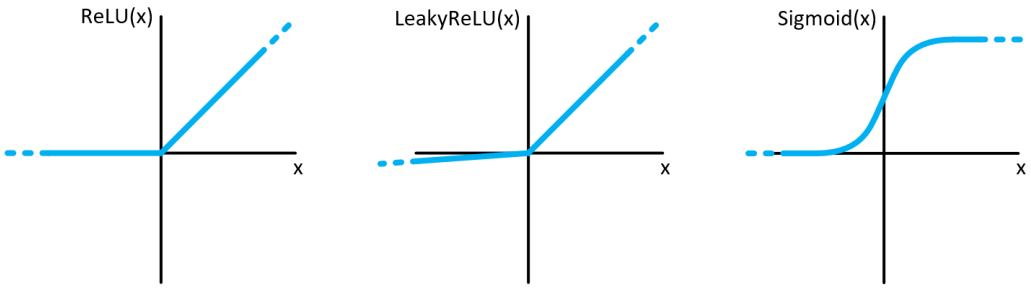


Figure 2.2: Different conventional activation functions. From left to right: ReLU, LeakyReLU, Sigmoid

Another common practice is the addition of a bias term to every hidden layer. This scalar bias term is a learnable weight which applies a shift to the result vector. Together with the bias term and the non-linear activation function the calculation of a hidden layer can be mathematically described as in Eq. 2.5

$$\mathbf{h}_i = f_i\left(\mathbf{h}_{i-1}\right) = \sigma\left(\mathbf{W}_i \cdot \mathbf{h}_{i-1} + b_i\right) \tag{2.5}$$

[1] The state of the vector does not hold contain any logical information which can easily be interpreted by a human and is only used as an in-between step by the NN to get to the desired output. Therefore it is referred to as a 'hidden' layer.

In the above Equation the i-th hidden layer of an MLP is described. When creating an MLP with n layers the full operation can be described as a composition of the above operation. This can be formally written down in function composition form as shown in Eq. 2.6. Where $\mathbf{q}$ describes the input and $\mathbf{v}$ describes the output.

$$\mathbf{v}(\mathbf{q}) = f_1\left(f_2\left(f_3\left(\ldots\right)\right)\right) = \left(f_1 \circ f_2 \circ \ldots \circ f_n\right)\left(\mathbf{q}\right) \tag{2.6}$$

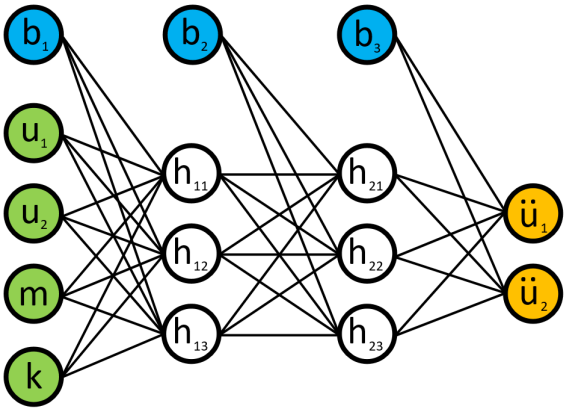


Figure 2.3: Proposed fully connected NN with 4 input features (green), 2 activated hidden layers of size 3 (white), bias-terms (blue), and output layer of size 2 (orange)

2.1.3. Training

Now we have a NN with two hidden layers and therefore three multiplication matrices. As mentioned before, we assume that there is a matrix $\mathbf{W_1}$, $\mathbf{W_2}$ and $\mathbf{W_3}$ for which performing procedure 2.1 results in the desired output $\mathbf{v}$. However, the contents of these matrices are as of yet unknown. To figure out these contents the network needs to be 'trained'.
Within this process the model is applied on data with a known outcome so that the difference between the actual and the desired output can be calculated. This difference, which is referred to as the 'loss', describes how well the applied NN fits to the trained data. It is the objective to minimize this loss in order to get the best performance[2]. For this minimization, the weight matrices can be updated by Gradient Descent. Where use is made of the gradient of the loss-value with respect to the weights, in order to find the new weights with a lower loss. When the data set becomes too large the loss is no longer calculated over the entire data set, and therefore the calculated gradient does not have to coincide with the 'actual' gradient of the entire set, which is illustrated in Figure 2.4. Every update based on a subset of the data is referred to as an 'iteration' and every run through the entire set is called an 'epoch'. For

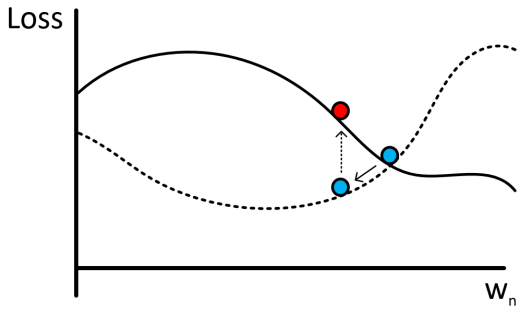


Figure 2.4: A gradient descent update based on a subset of the data (dotted line). The actual loss as function of the learnable weight given the entire dataset (solid line). The update moves the weight to a higher loss-value on the entire set (red dot).

[2]This generally holds true when the right loss function is chosen. In some cases it is not feasible to get a loss function which directly captures the desired performance. See section 3.4.4

the gradient descent update numerous techniques exist. In this report, the Adam optimizer is used for all weight updates (Kingma and Ba, 2017). This method uses a moving average of the first and second moment of the gradient of each weight in order to get an adaptive learning rate which is lower in case there is a high gradient variance of the given weight between batches. This is achieved by multiplying the learning rate with the first- and dividing it by the second moving moment.
As for the loss function there are limitless possibilities. Any numerical way to describe the performance where a lower value describes better performance can be used. It is however most common to use the mean squared error between the desired output or 'truth value' and the actual output.

2.1.4. Limitations of standard Neural Networks

Considering the multi-DOF problem above and the proposed solution scheme there are a few limitations to using regular NNs. These include lack of permutation invariance, the inability to generalise and a dependency between the number of learnable weights and the number of DOFs.
Firstly, recall we collected the features of the 2 DOF system as a vector:

$$\mathbf{q} = [u_1 \; u_2 \; m \; k]^T \tag{2.7}$$

After training, the physical behaviour of the given structure might be predicted sufficiently. However, in the case where the inputs are in any different order (permuted), the NN will need to train again since it is specifically designed for this permutation. Moreover, the addition of a degree of freedom or any other change within the space or relation between DOFs which is not covered in some way by relational input features will result in a need to train the network again. A trained network will not generalise to any such deviation between trained data and inference[3] data.
Lastly, recall that the size of the multiplication matrix connected to the input layer has the shape $\mathbb{R}^{n\times d}$. Where n was defined as the number of input features and d is defined as the size of the hidden layer. Due to this, the size of the learnable matrix will linearly increase with the number of DOFs. The same goes for the learnable matrix connected to the output layer. This means there is at least a linear relation between the number of DOFs and the number of learnable weights. Furthermore, an increase in DOFs will lead to an increase in complexity causing a need for bigger hidden layers. Therefore the relation between the number of DOFs and the number of learnable weights will be somewhere between linear and quadratic.

2.2. Graphs

A Graph $G = (V, E)$ is a set of nodes connected by edges where V is the set of nodes and E is the set of edges. Each edge is defined by two different points in the set of nodes.

$$E \subseteq \{\{i, j\} \mid i, j \in V \; and \; i \neq j\} \tag{2.8}$$

The connectivity of a discretized lattice structure can be described by a graph as shown in Figure 2.5. Every edge represents a connection between two nodes in the discretisation and the collection of all these nodes and edges encompasses the whole structure. It is important to note that the graph in itself does not describe the lattice structure within a coordinate space, but merely the relations between the nodes. The position in coordinate space can be defined as a feature attributed to the node.

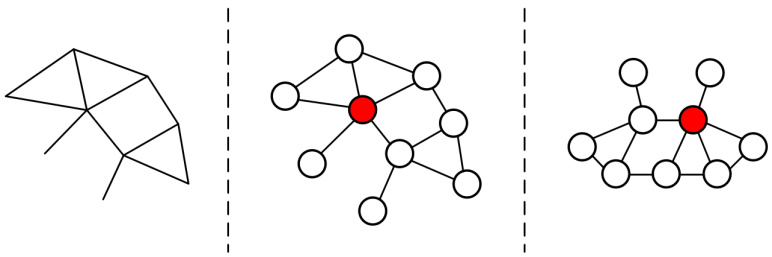


Figure 2.5: From left to right, schematized random lattice structure, graph of lattice structure, an identical graph with a different representation in space

[3] 'Inference' refers to the application of the network after training.

2.3. Graph Neural Networks

In paragraph 2.1 a very basic scheme was proposed to approximate the physical behaviour of a mass-spring system with two degrees of freedom. It was concluded that the use of regular NNs has concerns regarding permutational invariance, the ability to generalise and computational efficiency. Within the model, no use was made of the graph-like structure of these discretized problems where the node behaviour only depends on close neighbours in the graph space. By introducing Graph Neural Networks (GNNs) this relational inductive bias can be imposed on the problem (Battaglia et al., 2018). Furthermore, the behaviour of the individual nodes should be largely similar.
Within a GNN, messages get passed between nodes in a graph through message-passing layers. These messages can be composed based on the properties of nodes within the graph, or properties of the edges connecting the nodes. Within these layers, just like with regular NNs, there can be some set of learnable weights and some non-linear activation function. To elaborate on this let's look at the figure below. Which represents a more elaborate 2-dimensional multi-DOF mass-spring system.
For the following paragraphs, which explain some basics needed to understand the use of GNNs for the purpose of modelling dynamics, rich use has been made of the book by Hamilton, 2020.

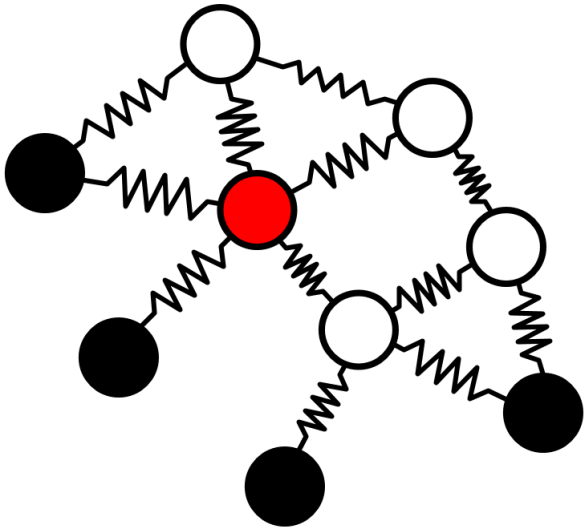


Figure 2.6: M-DOF mass-spring system with rigid nodes (black) and free nodes (red/white)

2.3.1. Message Passing Layer

Within this system, the acceleration of the red node depends on the exerted force of its neighbouring nodes.[4] This exerted force is a function of the relative displacements and the stiffness of the various springs. To pass this information from the neighbouring nodes to the red node, two steps are introduced. First, the embeddings[5] of the neighbouring nodes are aggregated. Second, the aggregated features are used to update the embedding of the node considered.
The aggregation of neighbouring nodes is done to retain permutational invariance. If there is some concatenation of the neighbouring nodes or edges, the order of input would again come into consideration. There are various possible aggregators, such as the sum, product and mean of the aggregated embeddings. Secondly, it needs to be considered whether to include the embedding of the evaluated node itself or not, which is referred to as a 'self-loop'.[6]
For the update function there is a wide variety of choices again. However, within the scope of this report, it is most interesting to look at an example where both the aggregated node embeddings and the embedding of the node considered are each multiplied by a learnable matrix $\mathbf{W}_{self} \in \mathbb{R}^{k\times k}$ and $\mathbf{W}_{neigh} \in \mathbb{R}^{k\times k}$ with k being the size of the node-embedding. Including some activation function this leads to the following computation for the message passing:

[4] This holds true under the assumption of a small enough time-step.
[5] The vector representing the nodes state is called the node-embedding.
[6] Also the way in which edge features are considered is part of the aggregation function, but this will be touched upon later.

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}\sum_{n\in\mathcal{N}(u)}\mathbf{h}_n^{(t-1)}\right) \tag{2.9}$$

This update function has to be applied to all nodes in the graph. While this might seem like a laborious task it can be simplified. Considering a matrix which includes all node-embeddings $\mathbf{H} = [h_0, h_1, ..., h_u]^T$ and the adjacency matrix which describes the graph[7] $\mathbf{A}$, Eq. 2.9 can be rewritten.

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{I}\mathbf{H}^{(t-1)}\mathbf{W}_{self} + \mathbf{A}\mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\right) \tag{2.10}$$

Applying the same function to every node in the graph enables the network to model different nodes with equal physical behaviour. Given equal input features and neighbours, any node in the graph will behave the same. Furthermore, the matrices which need to be trained do not scale with the size of the graph but only with the size of the node embedding. This illustrates another big advantage of choosing GNNs over regular NNs for discretized physics problems.

2.3.2. Edge features

So far only the features at the nodes were considered. However, in the physical discretization which was introduced earlier, the edges of the graph also hold valuable information. Within the mass-spring system, each spring might have a different stiffness or length for example. While these features could be averaged over the nodes to get an approximation, it makes much more sense to attribute these properties to the edges, which is done by introducing edge features.

The complication of edge features is that a way needs to be found to include them in the aggregate function. This could be done by first aggregating the embeddings of all the connecting edges. After which the aggregated edge embeddings are concatenated to the aggregated node embeddings. The new message-passing function then becomes:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}\left(\sum_{n\in\mathcal{N}(u)}\mathbf{h}_n^{(t-1)} \Big\| \sum_{m\in\mathcal{N}(u)}\mathbf{g}_m^{(t-1)}\right)\right) \tag{2.11}$$

Where $\mathbf{g}$ is the edge-embedding and the double separator symbol $\|$ indicates a concatenation. It is important to note that the dimension of the second weight matrix is different than before because of this concatenation $\mathbf{W}_{neigh} \in \mathbb{R}^{k\times(k+l)}$ with l being the size of the edge embedding.

The function above only updates the node embeddings. In some cases, it may also be useful to update the edge embedding. This could be done by, for each edge, concatenating the edge-embedding and the neighbouring node embeddings and multiplying by a trainable matrix $\mathbf{W}_{edge} \in \mathbb{R}^{(2k+l)\times l}$ as shown in Eq. 2.13. While it would be mathematically pleasing, it is not possible to combine both the edge and node update in one formula.

$$\mathbf{g}_u^{(t)} = \sigma\left(\left(\mathbf{g}_u^{(t-1)} \Big\| \sum_{n\in\mathcal{N}(u)}\mathbf{h}_n^{(t-1)}\right)\mathbf{W}_{edge}\right) \tag{2.12}$$

2.3.3. Addition of MLPs to GNNs

A GNN does not necessarily have to consist of only message-passing layers. One can add more general NN elements anywhere in the process. For example, the node features of a node itself could be multiplied by a matrix and modified by an activation function multiple times to form an MLP.

A use for this would be the 'encoding' of features. This way the NN could pre-process the node features into node embeddings. An important note here is that this can be done on a global level. This means that the weight matrix can directly be multiplied by a matrix containing all node embeddings like in Equation 2.10, but without taking into consideration the graph-like structure which is represented by the second term. An MLP encoder with 1 hidden layer and size n would look something like the following:

[7] The adjacency matrix is a matrix $\mathbb{N}^{n\times n}$ with n being the number of nodes in the graph, where each entry is set to 1 if the two indices describe an edge between two nodes, or a self-loop.

$$\mathbf{E} = \sigma\left(\mathbf{W}_2 \sigma\left(\mathbf{W}_1 \mathbf{H}\right)\right) \tag{2.13}$$

With $\mathbf{E} \in \mathbb{R}^{k \times n}$ representing the matrix of encoded node embeddings, $\mathbf{H} \in \mathbb{R}^{m \times n}$, $\mathbf{W}_1 \in \mathbb{R}^{k \times m}$, $\mathbf{W}_2 \in \mathbb{R}^{k \times k}$, n number of nodes in the graph, m is the number of original node features and k the desired size of the node embedding. If we look at the dimensions we again notice that the size of the learnable weight matrices is independent of the size of the graph itself.
A similar procedure could be followed after a message-passing layer to get from a (hidden) node embedding to an interpretable output (decoding), or in between message-passing layers. This will be further demonstrated in later sections.

3

Modelling Continuum Dynamics

Before considering the full beam behaviour in the next chapter, it is useful to first look into modelling one-dimensional continuum behaviour. This way it can be evaluated how a time-stepper GNN method performs for a simpler physical problem. In this chapter, a surrogate model is proposed to model the dynamics of the one-dimensional continuum. To simplify matters, only homogeneous initial value problems are considered.

3.1. Dynamic behaviour of 1D bars in extension

First, we will look into the conventional method to simulate the dynamics of a 1D bar in extension. To describe this behaviour the discretized Finite Element formulation (Wells, 2020) for dynamics in a continuum can be used:

$$\int_{\Omega_e} \mathbf{N}^T \rho \mathbf{N} \, d\Omega \, \ddot{\mathbf{a}}_e = -\int_{\Omega_e} \mathbf{B}^T \mathbf{D} \mathbf{B} \, d\Omega \, \mathbf{a}_e + \int_{\Omega_e} \mathbf{N}^T \mathbf{b} \, d\Omega + \int_{\Gamma_e} \mathbf{N}^T \mathbf{h} \, d\Gamma \tag{3.1}$$

In this chapter only homogeneous initial value problems are considered so the last two terms can be omitted. Further simplification to a 1D continuum leads to:

$$\int_{x_e} \mathbf{N}^T \rho_x \mathbf{N} \, dx \, \ddot{\mathbf{u}}_e = -\int_{x_e} \mathbf{B}^T \mathbf{D} \mathbf{B} \, dx \, \mathbf{u}_e \tag{3.2}$$

Where ρ_x is the mass per meter of the bar (kg/m), $\mathbf{u}_e$ is the discretized displacement field (m), and $\mathbf{D}$ is the constitutive matrix. Evaluating the integrals and rewriting into global form gives:

$$\mathbf{M} \ddot{\mathbf{u}}^t + \mathbf{K} \mathbf{u}^t = 0 \tag{3.3}$$

As can be noticed the dynamics of a homogeneous initial value problem is dependent on an equilibrium between acceleration, inertia and forces which get introduced by relative displacements. By finding a second coupling between acceleration and displacement we can get to a solution.
One approximation of this relation can be achieved by rewriting the central difference equation into an implicit form which yields:

$$\ddot{u}^t = \frac{u^{t-2} - 2u^{t-1} + u^t}{\Delta t^2} \tag{3.4}$$

Upon substitution and rearranging we get our fully discretized equation with which we can solve the problem at hand:

$$\mathbf{u}^t = \mathbf{M} \left(2\mathbf{u}^{t-1} - \mathbf{u}^{t-2}\right) \left(\mathbf{M} + \mathbf{K} \Delta t^2\right)^{-1} \tag{3.5}$$

3.2. GNNs as a substitute to simulate dynamic behaviour

An alternative method to simulate these problems would be through the use of GNNs. By learning a NN to predict the acceleration-field of a discretized continuum, based on its current kinematic state, we can omit Equation 3.1-3.3 from the process of calculating the dynamic behaviour. For linear calculations, there is no computational advantage other than avoiding matrix inversion, while for non-linear calculations it might be possible to avoid computationally elaborate schemes.

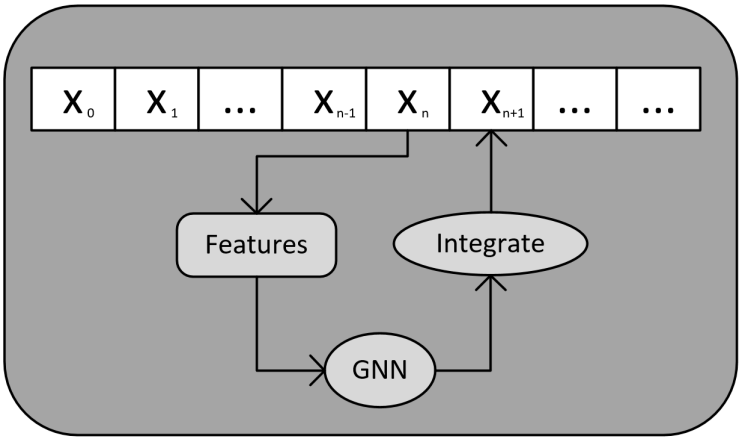


Figure 3.1: Outline of the time-stepper scheme to predict dynamic behaviour. x_n represents the kinematic state at time-step n

There is some physical intuition to performing this task with the use of GNNs. In the case we take a small enough time-step in a linear elastic system, we can state that the forces enacted on a node within the discretized system, are solely defined by the relative displacements to neighbouring nodes and some local material properties. By applying a single message passing step a GNN could theoretically accumulate these 'forces' to the node that is being evaluated. After which with the use of Newton's second law the acceleration within the node could be predicted.[1]

Another way to employ these time-stepper methods is by performing direct predictions where the GNN outputs the next step displacements. While such methods omit the need to integrate the result they also have major downsides in this context. It was found that models with direct predictions are highly unstable and show physically unrealistic inertial behaviour.

It was concluded that the difficulty in learning the kinematics of the continuum did not outweigh any conceivable advantage. Therefore this report solely focuses on an acceleration-based model.

[1] In practice the inner workings of a NN will not directly follow this physical intuition.

3.3. Demonstration of the concept

Let us consider a bar which is rigidly supported on both sides, with some initial velocity- and displacement field. The stiffness and density are uniform over the bar and do not need to be generalised, which means they can be left out of consideration[2]. The discretization will result in a non-uniform division[3] of the nodes over the length of the bar. Using the proposed scheme (Fig. 3.1) the dynamic behaviour of this bar will be predicted.

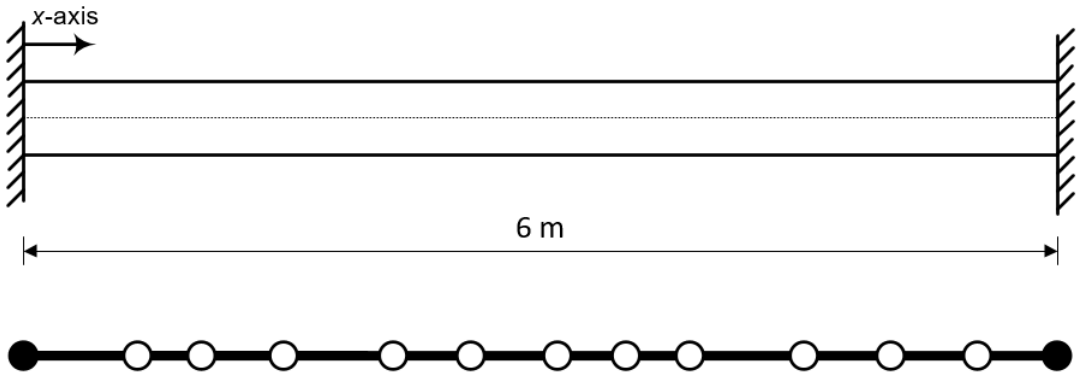


Figure 3.2: Bar to be analysed (top) and its discretization into nodes and edges (bottom). Discretization includes rigid nodes (black) and regular nodes (white).

3.3.1. input features

The outline of features which need to be used can be categorised as features to identify the forces, features to identify the inertia and features to identify the boundary. Since this is a discretized continuous system, the length of the edges (elements in FEM analogy) could be a good indication of the amount of inertia, this would need to be implemented as an edge feature. The magnitude of the forces is directly dependent on the strain, which will also be implemented as an edge feature.
Whether a node lies on the boundary can be indicated by a one-hot encoded vector[4] as a node feature. Since we are considering only zero Dirichlet boundary conditions the one-hot vector only needs a size of 2 to indicate whether a node is rigid or free[5].
Since the FEM solution presented above is used for training and makes use of an implicit formulation, the acceleration is dependent on one step of historical displacement. To make sure that the GNN theoretically has the same amount of information as the FEM scheme the nodes get the difference between the current and previous displacement as an extra feature for every node.

- Length of the edge as an edge feature (scalar)
- Strain of the edge as an edge feature (scalar)

- One-hot vector describing node-type as a node feature (size 2 vector)
- Previous displacement minus current displacement as a node feature (scalar)

[2]The NN will learn to fit the correct stiffness/mass-behaviour. Giving a constant value within the input features which is equal for every node in every sample has no effect on performance.
[3]The non-uniform division is chosen to enable generalization to continua of varying length.
[4]A one-hot vector is a conventional method to describe different node types. This is done by creating a zero vector, where every node corresponding to a given type gets a value of 1 at the same index of the one-hot vector.
[5]A single one-hot value would suffice in this case, but a vector is used for consistency.

3.3.2. GNN architecture

The architecture of the GNN is similar to that of Pfaff et al., 2020 and consists out of 3 separately identifiable parts. First, the features get encoded by an MLP with 2 hidden layers. Second, is an iterative process in which message passing occurs. This will be elaborated on in the next paragraph. Lastly, the node embeddings are decoded into the desired output by a 2 hidden layer MLP. with the desired output being the acceleration for every node. Every hidden layer is activated by a Leaky-ReLU function[6].

Within the iterative part of the algorithm, first the edge embeddings get updated by concatenating the current edge- to the neighbouring node embeddings and applying a 2 hidden layer MLP. Consequently, the updated edge embeddings are aggregated by summation and concatenated to the summed neighbouring node embeddings. This is followed by a 3 hidden layer MLP. This process is repeated a predefined number of times with every iteration having its own set of learnable weights[7]. Furthermore, in every iteration, the edge- and node embeddings get normalized by a Batch Normalization (Ioffe and Szegedy, 2015) after the first weighted addition[8].

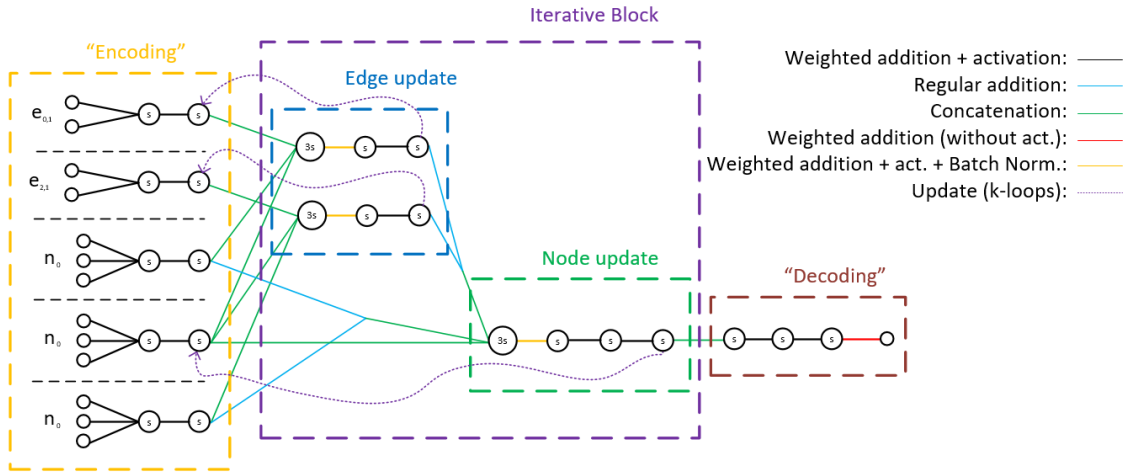


Figure 3.3: Architecture of the GNN with hidden layer size of s. The blocks 'Edge-update' and 'Node-update' together get updated a predefined number of times i.

3.3.3. Data-set

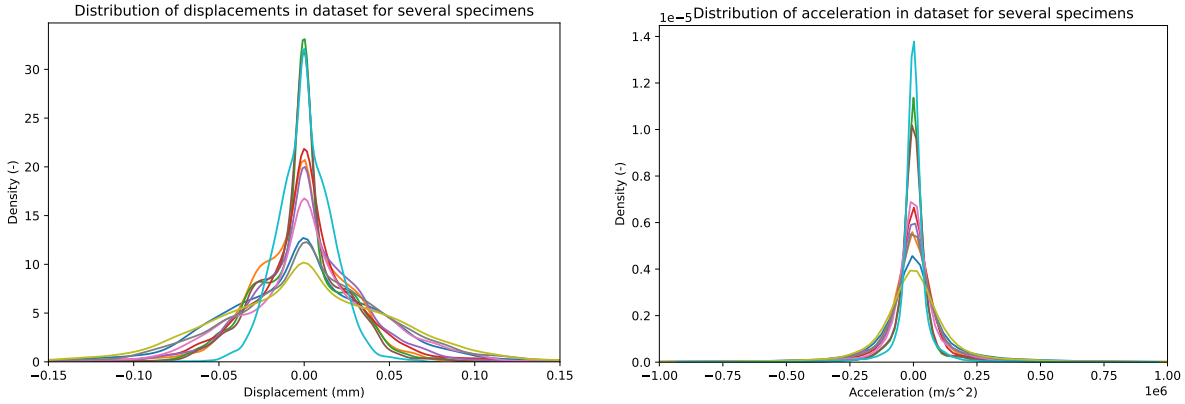
The train- and validation set is created by running the proposed FEM scheme on the same bar with various randomly created different initial conditions and meshes, while the time-step and physical properties of the bar remain unchanged. The GNN only needs to predict the acceleration based on the present kinematic information. Therefore each time step in each simulation is in essence a data point and is considered as such. These data points of all simulations are gathered, shuffled and saved to the data set. Apart from that, a zero-mean, unit variance normalization is performed on all features within the set. Furthermore, a split is made, allocating 80 percent of the entire data set for training, and 20 percent for validation. More details about the data set can be found in Appendix A.1.

The test set is created by applying the proposed FEM scheme on a problem with different initial conditions and double the specimen size to illustrate its capability to generalise.

[6] The Leaky-ReLU function was chosen since it might prevent the forming of 'dead' neurons.

[7] Tests were also performed on models with shared parameters between layers, but these performed significantly worse.

[8] The Addition of Batch Normalization led to considerable performance improvement and in some cases was vital to getting any result at all.



(a) (b)

Figure 3.4: Distribution of displacements and accelerations for a rollout of 10 different specimens

3.3.4. Training

Training is performed in batches of 512 time samples and an Adam optimizer is used with the standard parameters as described by Kingma and Ba, 2017. These are a learning-rate α of 0.001, β-parameters[9] of 0.9 and 0.999 respectively, no weight-decay and an ϵ[10] of $1e-8$. Training continues until no improvement of the validation loss has been reported for 50 epochs. It was found that longer training did not result in relevant improvement of the surrogate model. Because of the size of the data and GNN, the training ran on the DelftBlue cluster. ((DHPC), 2022)

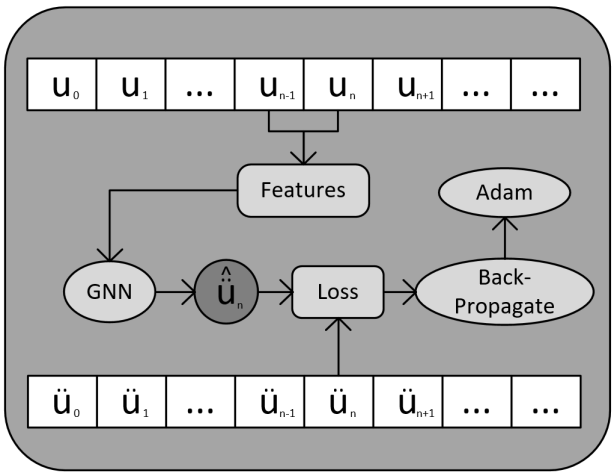


Figure 3.5: Visualisation of the steps taken to train the GNN

[9]The beta-parameters determine the rate of decay in the moving, first- and second-moment estimates. A higher value indicates a slower decay inducing a more gradual change in the moment estimates.
[10]Value added to the denominator to improve numerical stability. (Avoids division through zero in some cases)

3.4. Results

To show the performance of the proposed surrogate model upon inference various results are presented in the form of different evaluation metrics. After this, several architectural choices will be compared by changing hyper-parameters and plotting the performance of these various models. The variations include in chronological order: The addition of noise, variation in hidden-layer sizes and variation in the number of Message Passing Layers. Lastly, the oscillation shown in the evaluation metrics is investigated.

3.4.1. General Model Results

The first tests were performed using the default specifications as mentioned in Appendix A.3. Training was concluded after no improvement in validation error was recorded for 50 steps. The convergence of the model can be seen in Figure 3.6. The validation error does not have a trend which is diverging, so it is feasible that there is a potential to get even lower validation-loss values. However, as will be discussed later this wasn't found to have a heavy influence on the results.

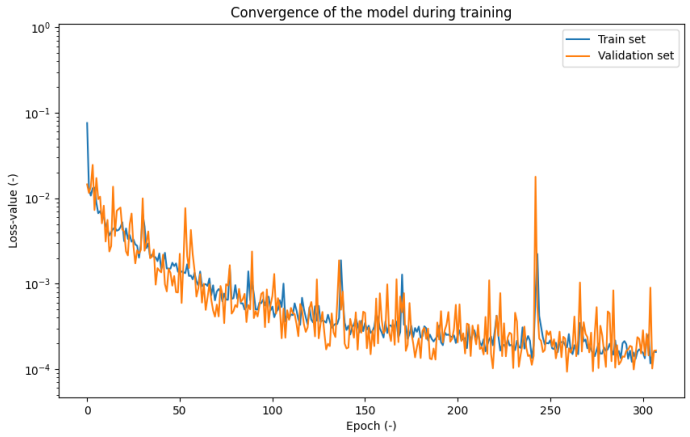
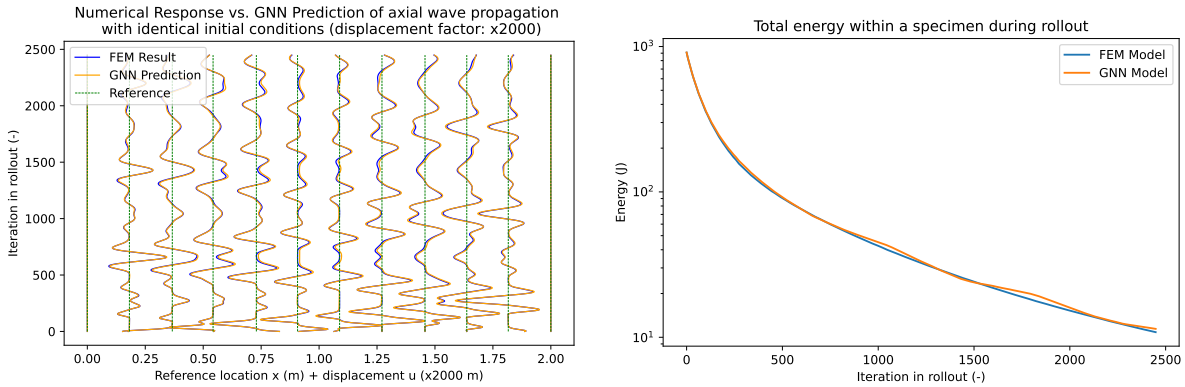


Figure 3.6: Convergence of the model when training with the default settings.

Figure 3.7a shows the displacements of a selection of nodes within the specimen over time as a consequence of both the FEM and GNN simulation. There are some slight deviations, but it is apparent that the wave speed and amplitude are roughly similar over the entire rollout. Secondly, looking at the energy over time within the system there is a decrease, which is imposed on the model due to numerical damping which arises from the implicit integration scheme used for the FEM calculation. This decrease also seems to be modelled by the surrogate, which is notable since it arises in a fully explicit forward-Euler GNN scheme. This indicates that the model is able to correct the true acceleration prediction to the value obtained by the implicit scheme.



(a) Displacement of every 9th node over time.

(b) Energy within the system for every step of the rollout

Figure 3.7: Behaviour of the GNN upon rollout for 3000 steps

Two long rollouts of the same trained network can be found in Appendix C.1. From these long rollouts it can be observed that as soon as the vibration takes the shape of the first eigenmode, the model under-predicts the frequency causing a drift between the FEM and Surrogate behaviour. One would expect this to be due to an under-prediction of the stiffness or an over-prediction of the mass. However, this should also result in a lower wave speed. The actual reason for this behaviour remains to be determined.

3.4.2. Qualification metrics

In order to quantify the quality of the network, three different qualification metrics are proposed. Firstly a displacement-based metric, secondly an acceleration-based metric and lastly an energy-based metric. This was done in order to analyse the performance of the surrogate model on the main physical properties of a simulation. Depending on the application these metrics might be valued differently.
All metrics perform a rollout on 20 specimens using 8 separately trained networks in order to get a reliable average. When different models get compared, the same specimens are used for each model within the comparison. The test specimens are two times as large as the train specimens unless stated otherwise. This is done to measure the capability to generalise to larger domains. The rollout starts from a randomly chosen point between iterations 50 and 1000 of a sample.

Displacement-based metric To quantify the predictive quality for exact displacements the Root Mean Squared Error (RMSE) is used. The error is calculated over the normalized displacement after i iterations. Where the normalization parameters get determined based on the displacements of the given sample within the entire rollout. Both equations combined bring about Eq. 3.9, which is referred to as the Normalized Root Mean Square Error (NRMSE). The purpose of the normalization is to quantify the erroneous deviation in the results relative to the deviation of the actual displacements.

$$NRMSE_{u;GNN} = \sqrt{\frac{\sum_{k=1}^{n}(u^i_{k_{GNN}} - u^i_{k_{FEM}})^2}{n \cdot \sigma\,(u_{FEM})^2}} \tag{3.6}$$

The NRMSE is continually calculated for all iterations in all rollouts and the average is taken per iteration. Performing the proposed scheme gives Figure 3.8 which shows the average displacement error as a function of the iteration within the rollout. It can be seen from the figure that the average error is not monotonically increasing and has an oscillatory component. While one would expect oscillatory effects within the error function of a single rollout, it is notable that after averaging multiple rollouts an oscillation persists. This can possibly be attributed to the frequency of the first eigenmode of the bar which is $8150\,rad/s$ or 771 iterations per oscillation with default time-increment[11]. Another explanation could be found in the wave speed which causes the wave to travel one bar length in 385 iterations. In section 3.4.8 it is found that the former is the most probable cause of this error.

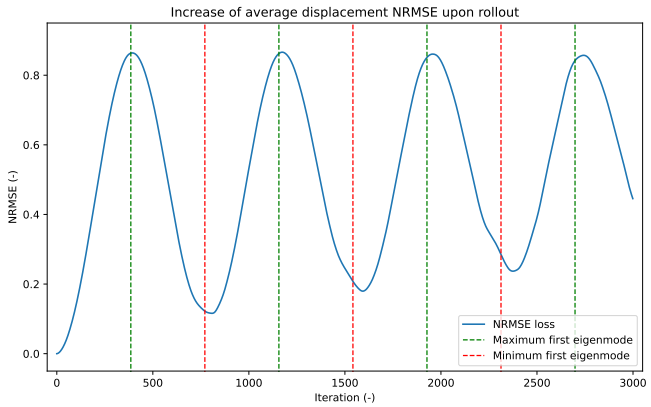


Figure 3.8: NRMSE error increase upon rollout for the default settings, with indicators for the analytical characteristic locations.

[11] A calculation can be found in Appendix A.5

It is important to recognise that, because of the above-mentioned, an NRMSE error at an arbitrarily chosen point does not have a good predictive value. It is proposed to look at the root-mean-squared error at the characteristic points of the oscillation. Therefore the possible evaluation points are defined as in Eq. 3.7 and Eq. 3.8. It should be noted that this method is not perfect since a slight difference is observed in the predicted versus the actual oscillation.

$$i_{eval;max} = \frac{(1+2n)\,\pi}{\omega \cdot \Delta t} \tag{3.7}$$

$$i_{eval;min} = \frac{2n\pi}{\omega \cdot \Delta t} \tag{3.8}$$

Acceleration-based metric The NRMSE can also be applied to the acceleration data of each iteration within a rollout. This has the advantage that oscillations in a low eigenmode do not significantly contribute to the acceleration and are therefore filtered out. While on the other hand, the waves with the highest displacement derivatives cause the highest accelerations which as a result are most prominent within this metric. The acceleration NRMSE upon rollout given the trained default model is shown in Figure 3.9

$$NRMSE_{\ddot{u};GNN} = \sqrt{\frac{\sum_{k=1}^{n}\left(\ddot{u}^{i}_{kGNN} - \ddot{u}^{i}_{k}\right)^{2}}{n \cdot \sigma\left(\ddot{u}\right)^{2}}} \tag{3.9}$$

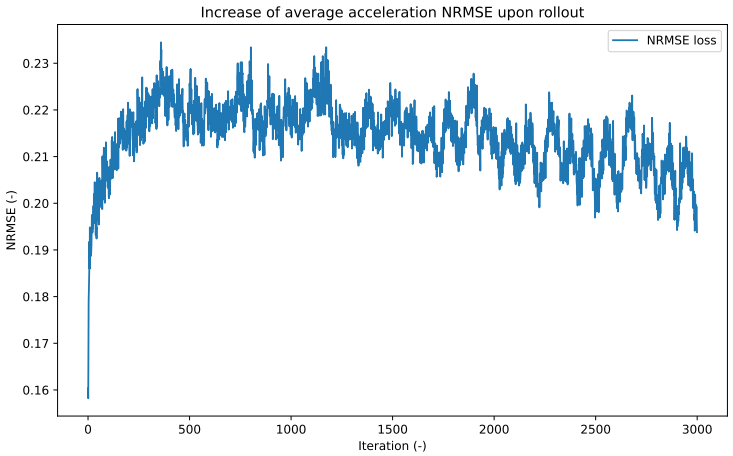


Figure 3.9: NRMSE error of the accelerations upon rollout with the default settings.

Energy-based metric The last proposed metric is the Energy Error Ratio (EER), which takes the difference in energy between the GNN and FEM model at an iteration within the rollout, and divides it by the energy in the specimen based on the FEM model. This results in Equation 3.10 where the energy function is defined as in Appendix A.4. This metric gives an indication of how well the explicit GNN scheme can predict the numerical damping which is a property of the implicit FEM scheme. The EER for the average of several runs of the default model is shown in Figure 3.10. It can again be noticed that also within the energy error there is an oscillatory behaviour.

$$EER_{GNN} = \frac{\left|\mathcal{E}\left(u_{GNN}\right) - \mathcal{E}\left(u_{FEM}\right)\right|}{\mathcal{E}\left(u_{FEM}\right)} \tag{3.10}$$

All of the presented metrics show useful information about the surrogate performance at inference. However, each of them might be valued differently depending on the intended application. The oscillatory error, for example, might not be a relevant issue when evaluating high-impact events given the

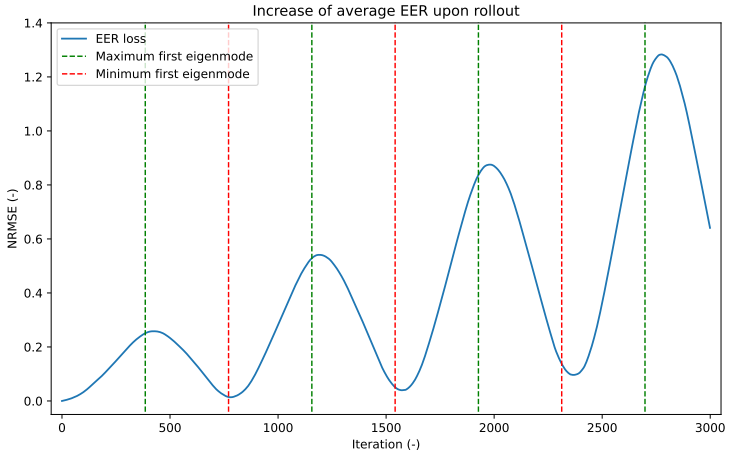


Figure 3.10: EER error increase upon rollout with the default settings.

comparatively small energy of these erroneous low-frequency vibrations. On the other hand, if there would be high interest in the amplitude or maximum deflection in the case of for example Serviceability Limit State design, the oscillatory error is detrimental to the quality of the surrogate given its objective.

3.4.3. 7-node demonstration

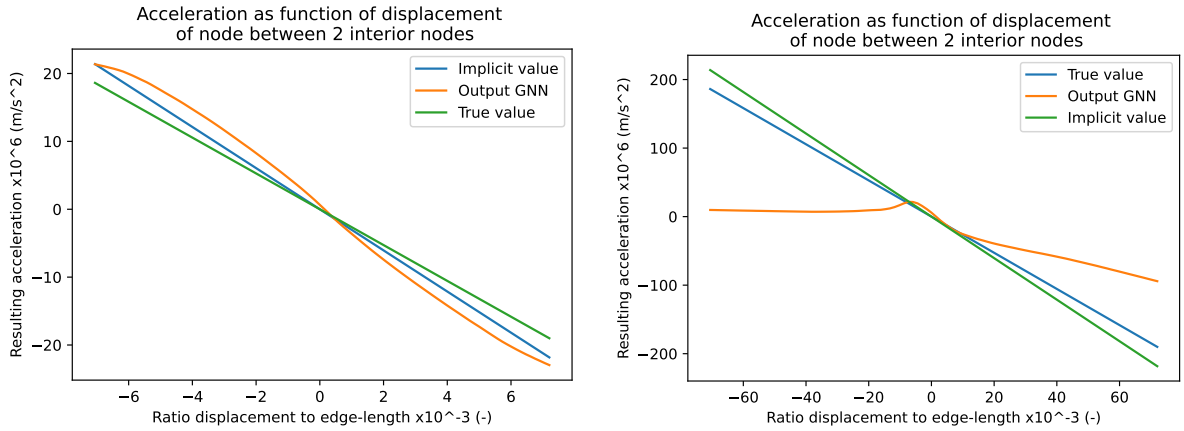
In order to better study the exact behaviour of the GNN, its output is plotted in a 7-node test. From a sample with 7 nodes only the middle is displaced by a distance u, consequently, this data is inserted into the trained GNN and the outcome is plotted[12]. This way it can be further investigated for which ranges of displacement the GNN functions and how well it approximates the FEM-based behaviour. Since no geometric non-linearities were given as training data it is not expected that these are learned by the GNN. Therefore in an ideal situation, the plot would give a straight line through the point (0, 0). The slope of the ideal line can be approximated by a discretized calculation. Assuming the mass at the node is equal to the element length and using the strains of the connected edges, one can find Eq. 3.11 through the application of elementary mechanics.

$$\ddot{u}_{approx} = -\frac{2 \cdot u \cdot E}{\rho \cdot l_{el}^2} \tag{3.11}$$

However, since we are dealing with an implicit calculation, the accelerations that result as an output from the FEM scheme are also dependent on the historical timestep. By using elementary mechanics and a bit of standard algebra the implicit-based acceleration can be calculated under the assumption of a zero-velocity field. This leads to Equation 3.12 which is derived in Appendix A.6.

$$\ddot{u}_{approx} = -u\frac{\rho}{2\frac{E\Delta t^4}{l_{el}^2} - \rho\Delta t^2} - \frac{u}{\Delta t^2} \tag{3.12}$$

[12]The choice of 7 nodes is made because in this scenario the information of the rigid boundary will not be passed onto the evaluated node (given 2 MPLs).



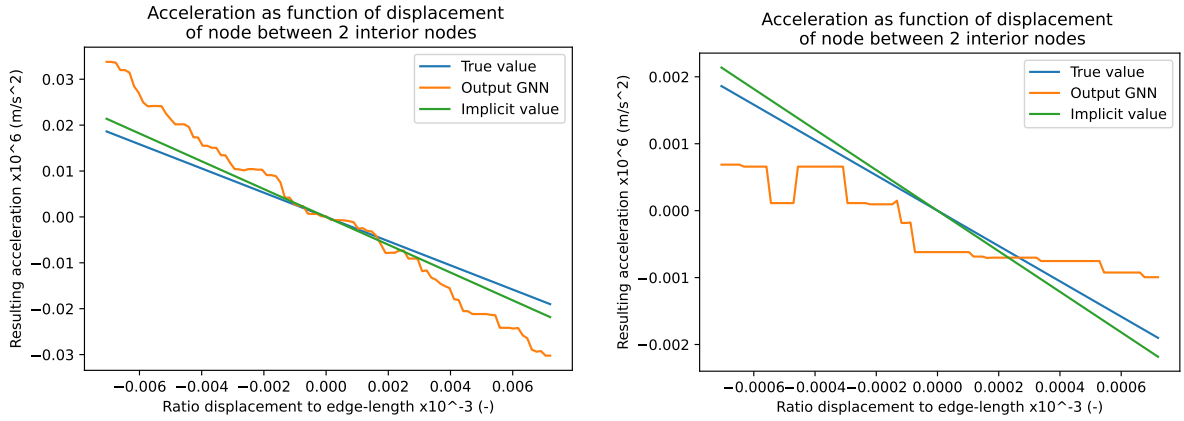
(a) Within trained data-distribution

(b) Outside of trained data-distribution

Figure 3.11: Predicted acceleration based on varying static initial displacement for a 7-node uniform specimen.

Figure 3.11 shows the 7-node test for the GNN trained with the default settings. The GNN seems to consistently overshoot the 'true' accelerations and does seem to cross through the point (0,0). It is notable that when the range of strains is increased beyond the strains seen in the training data, the model does not function (Fig. 3.11b). It can be concluded that to such values the model does not generalise well.

Judging Figure 3.11 one could make the assumption that the trained behaviour is a smooth polynomial-like approximation of the real behaviour. However, as Figure 3.12 shows, upon taking a smaller strain range it becomes visible that the approximation is not at all smooth. Considering the inner functioning of the NN this is quite logical. The standard MLP structure is theoretically unable to perform a perfect multiplication of the physical inputs to give a generalized result like Eq. 3.11 or Eq. 3.12. Any hidden layer only communicates with the next layer through a linear matrix multiplication which results in the capability to sum. The activation functions, in turn enable the network to approximate any function. However, the Leaky-ReLU activation function used for the presented model, inherently creates discontinuities given its discontinuous nature.



(a)

(b)

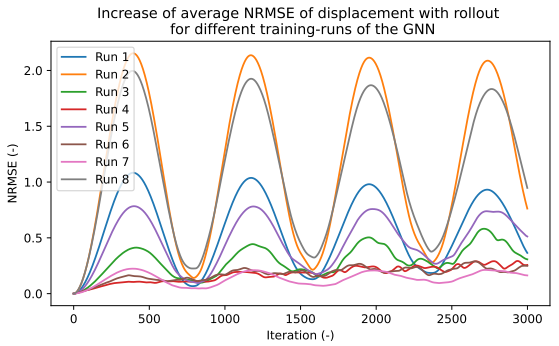
Figure 3.12: Predicted acceleration based on varying static initial displacement on a small range for a 7-node specimen.

3.4.4. Variation between trained models

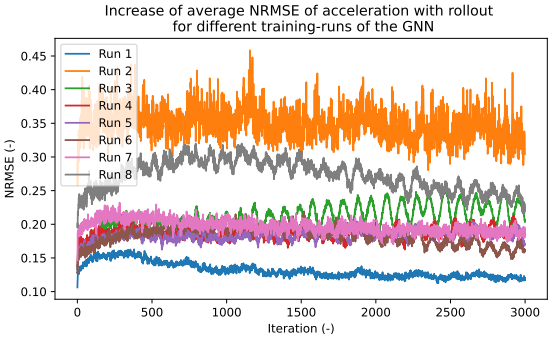
Table 3.1 shows the number of epochs[13] and the best validation loss found for 8 different training runs on the same model. It shows a variation within the validation loss of around 50 percent around the mean which is a comparatively small difference considering the orders of magnitude. To compare the results from these runs Figure 3.13 shows the respective performance on the proposed metrics. What stands out in this figure is the substantial variability between these trained models.
The observed variation creates an extra challenge regarding the quantification of network performance. A way to deal with this would be to take the trained network with the lowest validation loss. However, upon closer inspection of the training-run performance compared to the final validation loss only a weak correlation can be observed. Examples of this are training run 2 with an average loss and the objectively worst performance and run 5 with the second-best loss and average performance.

Training-run	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8
Total epochs	328	280	197	308	340	280	408	179
Validation loss ($x10^{-4}$)	1.0239	1.0704	1.5250	0.9332	0.7528	1.2795	0.6561	1.5972

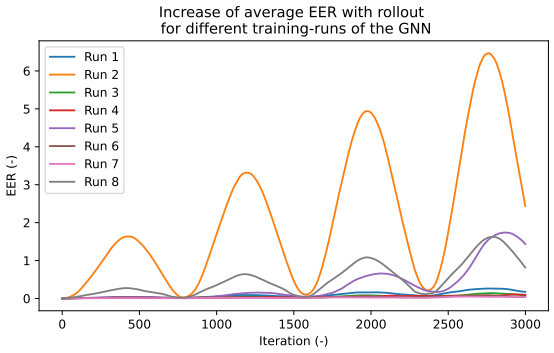
Table 3.1: Training loss and epochs for all different displayed runs on the same model



(a) NRMSE of displacements



(b) NRMSE of accelerations



(c) EER

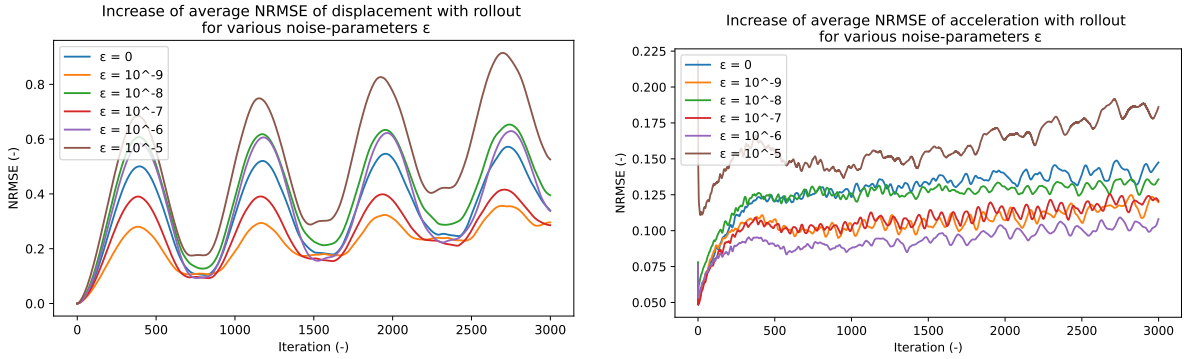
Figure 3.13: Performance of the model for 8 different training runs

There is no direct correlation between performance within any of the three metrics either. Taking a closer look at the rollout of the model based on the first training run it can be noted that, while having a remarkable acceleration accuracy, there is a sizable oscillatory error which results in average to bad performance in the displacement and energy metric.
These different positives and negatives make it hard to choose a 'best' model in general. Furthermore, given the large variation, lots of trained networks are needed to lower the impact of a lucky shot on drawing the right conclusions. Therefore it is chosen to take the average of all trained models when the performance is evaluated. As a result, upon training several networks, the optimal performance will always be better than the displayed result.

[13] The length of each run varies because training is terminated after a lack of improvement on the validation set for 50 epochs.

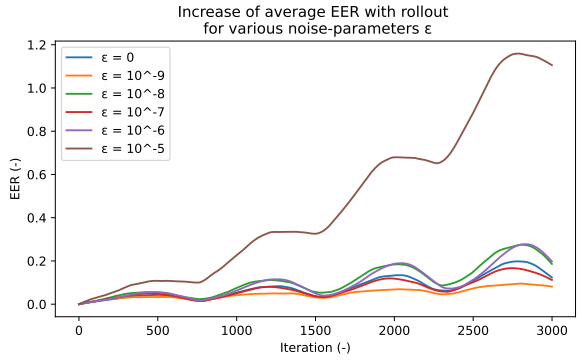
3.4.5. Introduction of noise

Noise was introduced to the training set using the algorithm described in Appendix A.2. The noise can be adjusted through two hyperparameters, one which describes the magnitude of the noise and one which describes the ratio between correction for velocity vs. correction for displacement. Further specifications of the latter can be found in the appendix and in the paper by Pfaff et al., 2020. To investigate the use of noise within the context of the 1-dimensional GNN model, it was trained on various values of the magnitude parameter ε. A comparison of the average behaviour upon rollout can be seen in Figure 3.14. From tests it appeared that bigger values than the displayed noise range (10^{-5}) gave large incremental errors, so these are not displayed.



(a) NRMSE of displacements

(b) NRMSE of accelerations



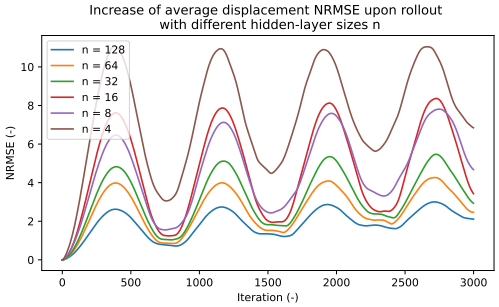
(c) EER

Figure 3.14: Performance of the model for different noise hyper-parameters.

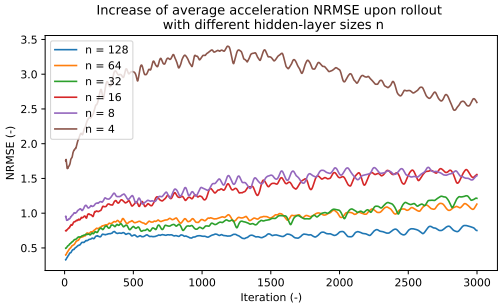
Comparing the different noise values the model results are quite similar. When considering the magnitude of the error induced by the oscillatory part some noised models seem to perform slightly better. However, no linear correlation can be found between the value of the noise parameter for the range $[10^{-6}, 10^{-9}]$ and the performance in displacement and energy error. Taking the average performance of the noise values in this range it is comparable to the zero-noise case. Considering the lack of correlation and the small performance difference it could be argued that the observed performance improvement is due to the variation between trained models (section 3.4.4). Therefore it cannot be directly stated that the applied noise has a positive influence.
The increase of the acceleration error shows a slightly better average performance of the models trained with noise. However, for a more definite result the number of trained models per test, which is 8, should be increased.

3.4.6. Different layer-sizes

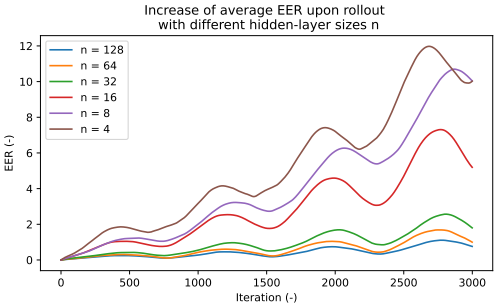
To investigate the effect of the number of neurons per layer a test was performed for hidden-layer sizes with different powers of 2. For each size, 8 models were trained and rolled out according to the methods described before in order to evaluate the average performance. Figure 3.15 shows the performance of these models on the proposed metrics. What can be clearly seen in this figure is the strong correlation between the number of neurons per layer and performance. Moreover, the observed correlation is displayed for both the oscillatory error and the general error increase at the characteristic points.
Taking the average of the 8 trained models it is obvious that a hidden layer size equal to 128 gives the best result. Given the strong correlation and the magnitude of the performance difference, it can contrary to the previous section be stated that the models with more neurons perform better on average. On the other hand, it is not directly possible to eliminate the other layer sizes as bad performing. Only the layer size of 4 gives a notably worse result in the 'actual' displacement error increase at the characteristic points as well as the acceleration error.



(a) NRMSE of displacements



(b) NRMSE of accelerations

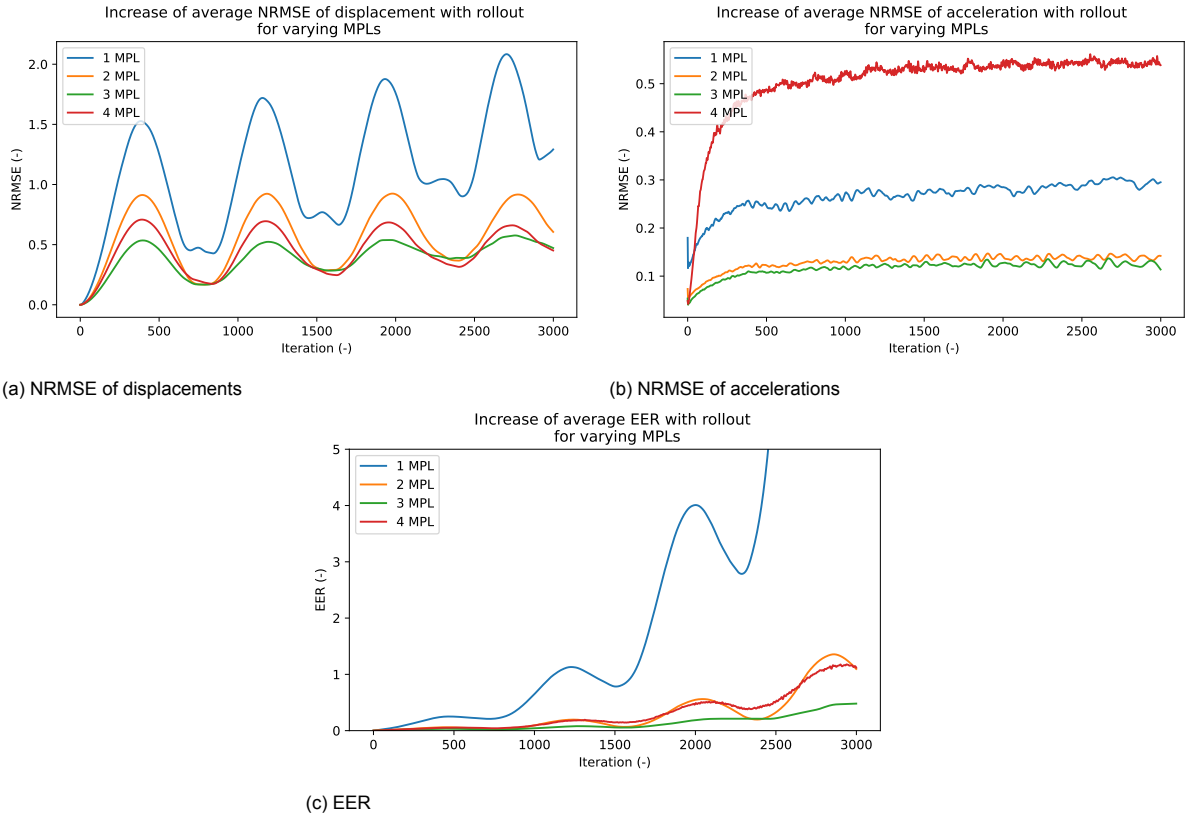


(c) EER

Figure 3.15: Performance of the model for different layer sizes.

3.4.7. Multiple MPL

A comparison of the performance upon a variation of the number of Message Passing Layers can be found in Figure 3.16. As it appears the model with only one MPL is consistently worse, both in the oscillatory error and in the general error growth at the characteristic points. A reason for this could be the combination of the implicitly generated data and the inability to correct the estimation based on just one MPL. Within implicit integration methods, the integration result is directly dependent on itself. A way to deal with these kinds of mathematical formulations is by using an iterative scheme where the result is approximated. While it cannot be proven, it is conceivable that the GNN uses some sort of iterative prediction scheme to deal with this implicity.
The model with 3 message-passing layers performs best considering the oscillatory error while having comparable performance to the 2 MPL model at the oscillation minima and within the acceleration error. It cannot definitively be concluded whether 3 MPLs is better than 2 because of the variation between trained models (section 3.4.4).
From the evaluation-metric plots it also becomes apparent that the 4 MPL-based model shows some noise-like behaviour or an oscillation on a very high frequency. This behaviour is nonphysical and the model is therefore considered worse. For larger MPL sizes the performance deteriorated further.



(a) NRMSE of displacements

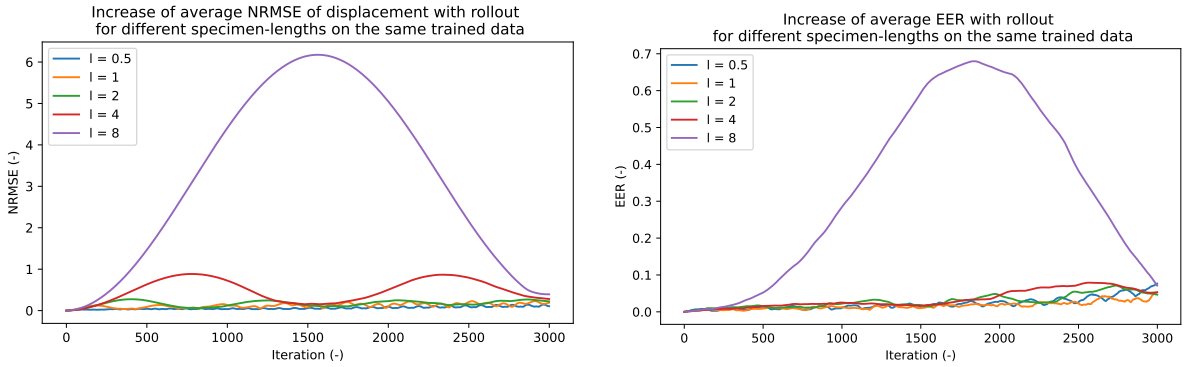
(b) NRMSE of accelerations

(c) EER

Figure 3.16: Performance of the model for different amounts of Message Passing Layers.

3.4.8. Oscillating error

In section 3.4.2 it was mentioned that the period of the oscillating error upon roll-out was equal to the period of the first eigenmode. This oscillation also coincides with the time it takes for a wave to travel through the entire continuum. To investigate what is the exact root cause of this oscillatory error and to confirm that this relation holds for different specimen sizes some figures are shown.
Firstly, to confirm that this effect holds for bigger domains, the model was rolled out on specimens with sizes which differ from the specimens in the training data by a factor of two. The results from this test are shown in Figure 3.17. The figure clearly shows that for each doubling in beam length the period of the oscillatory error also doubles. This proves that this error is either induced by an eigenmode, or by some effect arising from the wave speed.



(a) NRMSE of displacement upon rollout

(b) EER upon rollout

Figure 3.17: Behaviour upon rollout for different specimen lengths on a model trained by data where specimen-length = 1

Secondly, to determine the root cause, it is interesting to look at the behaviour of a specimen with trivial initial conditions (Fig. 3.18). This test was performed on a trained model with the default hyper-parameters. The test shows that even for zero-initial conditions the oscillatory error gets induced[14]. A probable cause for this erroneous introduction of movement can be found in Figure 3.12b. In a zero velocity-, zero displacement field the Surrogate model still predicts a small acceleration. This acceleration is a uniformly introduced error over the entire specimen which results in a vibration in the first eigenmode.

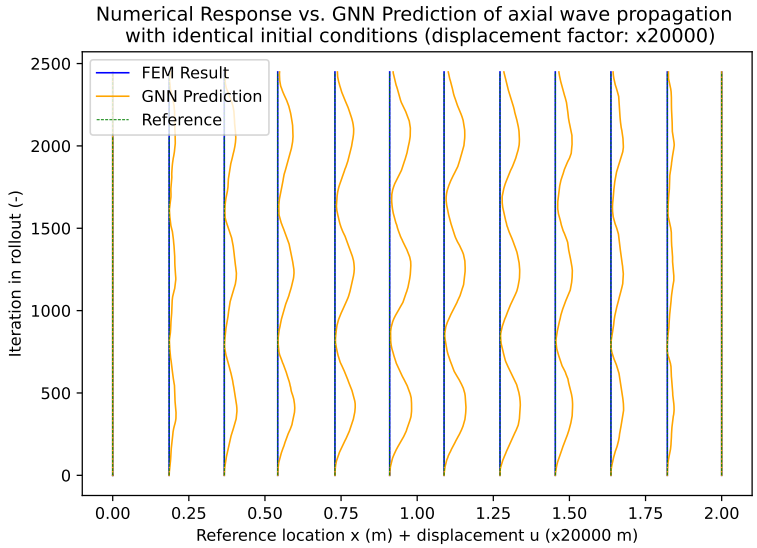


Figure 3.18: Propagating error for a model with zero initial conditions

[14] The scale of this oscillation is considerably smaller than that of the waves introduced in for example Fig. 3.7a

3.5. Discussion

The proposed surrogate model in this chapter proves to be capable of simulating the dynamics of a one-dimensional beam with long-term stability. Furthermore, it can effectively generalise to larger continua. The most prominent drawback of the model is the erroneous introduction of an eigenmode vibration in almost all simulations. Considering the ultimate goal of the surrogate is not to predict the FEM solution, but the real-world behaviour of such a continuum this introduced error is not deemed critical since it is handled in a physically correct way.

The surrogate is not able to generalise to displacement values outside of the trained range. For further research, it can be interesting to evaluate the relation between the trained range and the performance on sub-ranges with different magnitude scales.

A last important remark is that different training runs using the same parameters do not lead to the same performance. Furthermore, it is not directly possible to classify which of the trained models performs best since there is not always a model which performs best on all metrics. It was also noted that there is only a weak correlation between the single-step prediction error and the rollout performance.

4

Modelling Beam Dynamics

In the previous chapter, a surrogate model for simulating the dynamic behaviour of a 1-dimensional continuum was presented. In order to simulate the behaviour of lattices, there is a need to model as well the extensional as the bending behaviour of the lattice elements. This chapter introduces a surrogate model for beam dynamics, where single geometrically linear Timoshenko beams are modelled in a two-dimensional space. Within this chapter, firstly the analytical formulation of the dynamics of a beam will be investigated in order to get an intuition for the behaviour that needs to be modelled. Secondly, the proposed surrogate model is elaborated and lastly, the results are presented together with some investigations into improvements of the model.

4.1. Timoshenko Beam Theory

To understand the behaviour of a beam, we will take a close look at what happens within a small segment of a static beam. As can be seen in Fig. 4.1, three different section forces are present within an idealized beam (normal-, shear- and moment), which all deform the given beam section in a different way. The normal force causes a direct elongation of the beam part. The shear force causes a distortion where the cross-sections move parallel to each other. And the moment force causes a rotation of the section, which is a consequence of the lower 'fibres' stretching, and the upper fibres contracting.

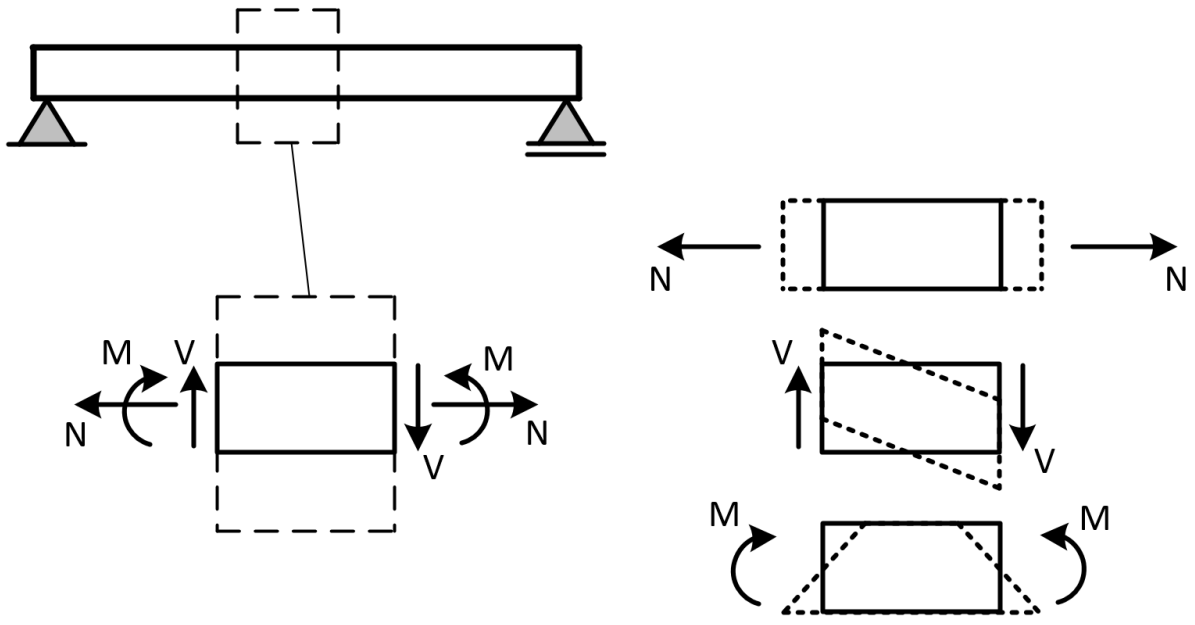


Figure 4.1: Deformation of a small segment of a beam due to section forces.

While the illustration above gives some intuition into the local behaviour within a beam segment, it does not enable us to evaluate the deformation of the entire beam. In order to relate these local strains to displacements on a global scale a differential equation has to be derived. The two most prominent theories, which found such derivation, are the Euler-Bernouilli and the Timoshenko beam theory.
Firstly, the Euler-Bernouilli-model assumes that every cross-section remains perpendicular to the curve of deflection (Euler, 1744)[1]. Through a change in rotation of the cross-section (curvature), a gradient in elongation will occur perpendicular to the rotation axis. The capacity to take up moment forces is a direct cause of this gradient of the (fibre-)elongation and thus the curvature of the deflection line. Within this theory, solely the moment distribution causes displacements in the form of beam rotations. Due to the assumption that the sections remain perpendicular, the beam rotations can be integrated along the beam in order to get the deflection line. A more elaborate explanation of the theory can be found in many structural engineering books, like Hartsuijker and Welleman, 2016.
Before moving on to the Timoshenko beam theory it is interesting to look at a beam with only shear deformation. Here the cross-sections do not remain perpendicular to the deflection curve but remain parallel to each other. The ensuing deflection is no longer a cause of the elongation of fibres but of the shearing of the cross-sections. It is important to see the distinction that, within the pure shear beam, the 'sections remain perpendicular' assumption is made in order to model pure shear behaviour but is not a restriction for shear deformation to occur[2].
Both physical phenomena are present in the behaviour of a regular beam, but in many cases, one of the two can be neglected. For example, when the bending stiffness is much larger than the shear stiffness, the deflection will be governed by the latter, since there will be almost no change in the rotation of the sections. The opposite goes for a larger shear stiffness with a smaller bending stiffness. If either of the phenomena-related stiffnesses goes to infinite the deflection will be fully governed by the other. Therefore we can conclude that the combination of the two phenomena forms a typical series system.

$$k_\gamma = GA_s = \frac{E}{2(1+\nu)} bh\,\kappa_1 \tag{4.1}$$

$$k_\phi = EI = E\frac{bh^3}{12} \tag{4.2}$$

The equations above (Eq. 4.1 & 4.2) show the stiffness for shear deformation and bending deformation respectively given a homogeneous isotropic rectangular cross-section. Where E is the Youngs-modulus, G the shear modulus, ν the Poisson's ratio, b section width, h the section height and κ_1 the correction factor for the shear-surface which is 0.8333 for rectangular sections (Freund and Karakoc, 2016). Both the shear- and bending-stiffness are dependent on the height of the cross-section. However, the former increases linearly with the height, and the latter with a third-order relation. Therefore, in the case of very slender[3] beams, the bending behaviour is dominant to determine deflections and for very thick beams the shear behaviour is. In practice, it may be decided to model only one of the two behaviours if the effect of the other is negligible.
In the case of transverse wave-propagation within beams the need to model either of the processes is not, like in the static case, dependent on the ratio between the height and length of a beam, but on the wavelength.
Since there is an infinite amount of perceivable wavelengths in a beam, there is a need to model both physical phenomena simultaneously. To do this, the differential equations proposed by Timoshenko (Timoshenko, 1921) can be used. His model adapts the Euler-Bernouilli theory by adding an extra rotation variable which describes the section rotation as shown in Figure 4.2. The ensuing relation between the derivative of the deflection, the cross-section angle and the shear angle, assuming small rotations, is given in Eq. 4.3.

$$\frac{\partial w}{\partial x} = -\phi - \gamma \tag{4.3}$$

[1] The cross-sections remains perpendicular assumption was made by Jacob Bernouilli, as was the discovery that the curvature is proportional to section-moment. (Timoshenko, 1953)
[2] If the sections do not remain parallel, a change in rotation is observed which causes Bernouilli bending behaviour. Therefore it would not be pure shear.
[3] The degree of slenderness describes the ratio of height versus length of a beam.

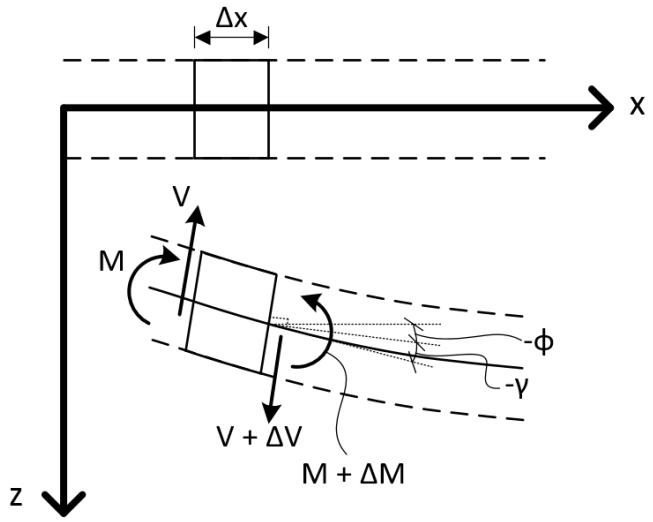


Figure 4.2: Element-definition for a Timoshenko beam-element.

Considering that the shear force is directly dependent on the shear deformation and stiffness, while the moment is directly dependent on the change in rotation and bending stiffness, the constitutive relations 4.4 and 4.5 can be constructed. The equations of motion, which follow from the defined section forces on the differential element and Newton's second law, can be formulated as in Eq. 4.6 and 4.7[4].

$$V = -GA\,\gamma = GA\left(\frac{\partial w}{\partial x} + \phi\right) \tag{4.4}$$

$$M = EI\,\frac{\partial \phi}{\partial x} \tag{4.5}$$

$$\frac{\partial M}{\partial x}dx - V dx = \rho I \frac{\partial^2 \phi}{\partial t^2}dx \tag{4.6}$$

$$\frac{\partial V}{\partial x}dx = \rho A \frac{\partial^2 w}{\partial t^2}dx \tag{4.7}$$

By substituting the constitutive into the dynamic relations, a system of two coupled partial differential equations is acquired with the variables ϕ and w.

$$EI\,\frac{\partial^2 \phi}{\partial x^2} - \rho I \frac{\partial^2 \phi}{\partial t^2} - GA\left(\frac{\partial w}{\partial x} + \phi\right) = 0 \tag{4.8}$$

$$GA\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial \phi}{\partial x}\right) - \rho A \frac{\partial^2 w}{\partial t^2} = 0 \tag{4.9}$$

The shown PDEs model the lateral displacement of a Timoshenko beam. Since geometrically linear behaviour is considered, the bending and extension deformation are decoupled. Therefore the axial behaviour is defined by a third fully decoupled PDE shown in Eq. 4.10. The equations shown in this section are purely based on the local beam definition. Within the global space, the displacements will be coupled, unless the beam coincides with the global axes. By applying a rotation matrix, constructed using the beam angle relative to the global coordinate system, one can get the local decoupled displacements.

$$EA\frac{\partial^2 u}{\partial x^2} - \rho A \frac{\partial^2 u}{\partial t^2} = 0 \tag{4.10}$$

[4]Important to note is that the rotatory inertia of the cross-section is also taken into account in this formulation (Eq. 4.6). This is an addition within the Timoshenko theory which is not modelled in Bernouilli beam theory.

4.2. GNN Surrogate Model

To create a NN-model which acts as a surrogate to simulate combined bending and extension the same general architecture as in Chapter 3 is used. However, there are some modifications needed within the in- and output and the training data.

4.2.1. Input- and output-features

Firstly, considering the theoretical derivation shown before there are three variables which describe the beam behaviour. These are the displacement normal to the beam-direction u, the displacement lateral to the beam-direction w and the rotation of the cross-section ϕ. Within a problem where multiple beams intersect each other, it is not possible to directly take these variables as the DOFs for each node, since at the beam intersection the normal cannot be defined. Therefore the predicted node accelerations are modelled in the global x- and z-direction[5]. For the section rotation ϕ, the 'local' variable can still be used since this value is shared by all connecting beams given the rigid connections.
Considering that the internal forces within the beam are dependent on the local strains, it is chosen to give the locally defined strains over each edge as an input feature. Similar to the 1D continuum case the local axial strain is defined as an edge feature. For the local lateral strain, which has an immediate effect on the internal shear, the strain is defined in the same way. The internal moment forces are directly dependent on the curvature which is the derivative of the section rotation ϕ. Therefore this derivative, which could be seen as the rotational strain, is also implemented as an edge feature. The general form of the three edge strains defined above is shown in Eq. 4.11, where χ can be substituted with any of the local variables u, w and ϕ.

$$\frac{\partial \chi}{\partial s} \approx \frac{\chi_{n+1} - \chi_n}{\Delta s} \tag{4.11}$$

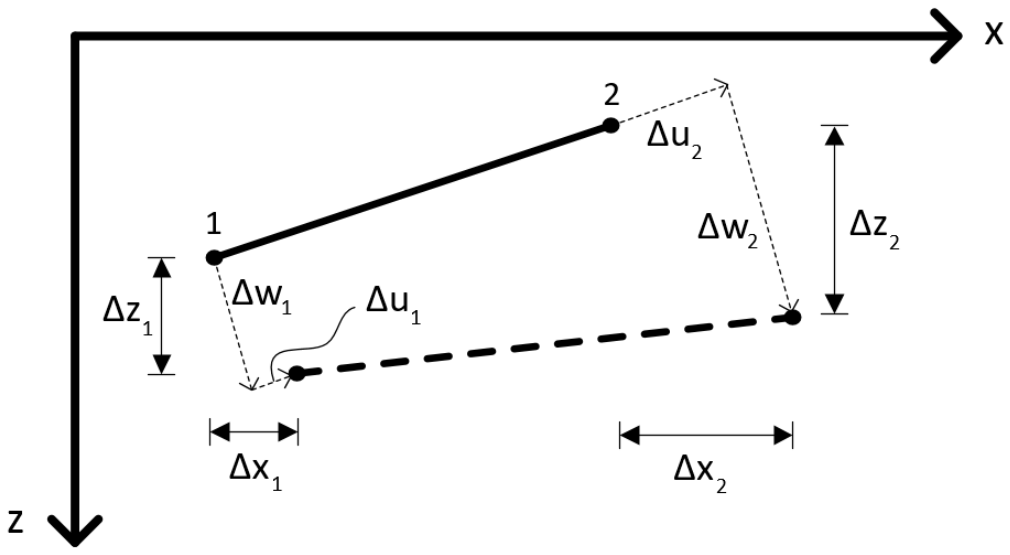


Figure 4.3: Definition of local and global displacements of an edge between two nodes.

Given the proposed edge features it is not directly possible to calculate the shear strain since this is both dependent on the lateral strain and the section rotation (Eq. 4.3). Therefore, the section rotation is also given as a node feature.
With the total of these four features, the network could theoretically describe all local section forces. However, the desired output is in the global coordinate system. For the network to transpose the local behaviour into global outputs, the vector of the edge is given as an edge feature. This edge vector also describes the length of the edge which gives the network the required information to determine the inertia.

[5]For the example with one beam, the local directions could still be used. However, the global directions are chosen to stay consistent with the end goal.

Lastly, just like with the extension case a one-hot vector describes whether a node is free or restrained. Furthermore, the difference of the DOFs between the two previous time steps is given.

Edge-features:

- Vector of the edge (size 2 vector)
- Longitudinal and lateral 'strain' of the edge (size 2 vector)
- Rotation-derivative linearised over the edge (scalar)

Node-features:

- Rotation ϕ of the node (scalar)
- One-hot vector describing node-type as a node feature (size 2 vector)
- Previous displacement minus current displacement as a node feature (size 2 vector)

Output:

- Acceleration in global space for every node ($\ddot{x}$ and $\ddot{z}$)
- Angular acceleration of the section at every node ($\ddot{\phi}$)

4.2.2. Data-set

In order to train the GNN to predict the dynamics of a single beam in any configuration, 40 randomly chosen beam configurations with randomly generated initial conditions have been constructed. These specimens are rolled out for 10000 steps using a FEM-solution procedure and at each step, the kinematic information is saved. The FEM scheme makes use of the Adams-Bashforth 2-step integration method (Bashforth and Adams, 1883). The acceleration at each separate timestep of each specimen is taken as a training point with the exception of the first 100 iterations. This is done to get less steep displacement gradients within the data set since the initial conditions are not smooth. Hyper-parameters and specimen properties are further listed in Appendix B.2.
The angle of the beam θ is varied in the range $[0, \pi]$ in order to train a model that generalizes to any angle[6]. The length of the beam is 1 mm for every sample and has a uniform mesh with 11 nodes. The initial conditions are randomly generated by the procedure elaborated in Appendix B.1.

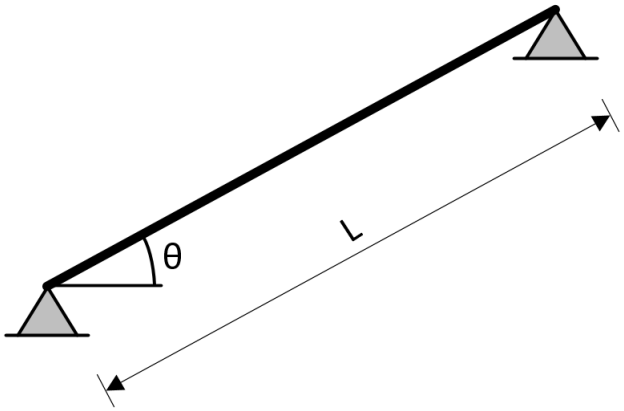
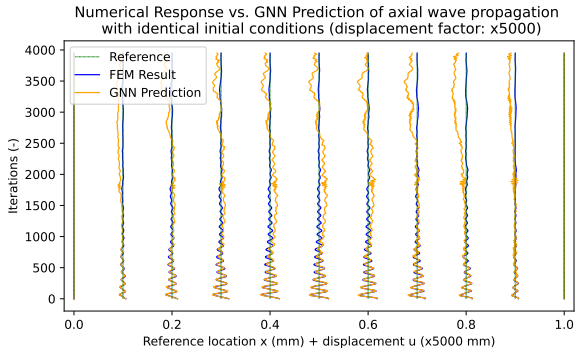


Figure 4.4: Generic specimen in data set with random beam-angle θ

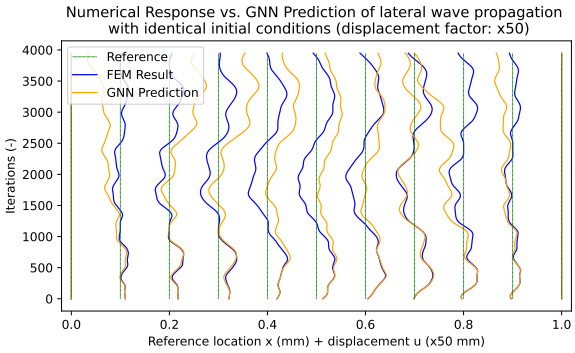
[6]All edges are defined in both directions, therefore the angles in the range $[\pi, 2\pi]$ are automatically modelled as well.

4.3. Model Results

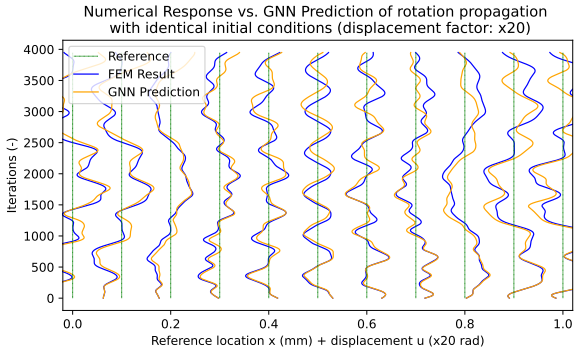
When training the proposed model, no long-term stable results could be achieved. As Figure 4.5 shows, the lateral degree of freedom, which is influenced by the bending and shear behaviour, already differs significantly from the true solution after the first 2000 iterations. While a well-performing rollout for 2000 steps may seem like a lot, it is just enough time to model half of the period of the first eigenmode vibration.



(a) Axial displacements over time



(b) Lateral displacements over time



(c) Rotations over time

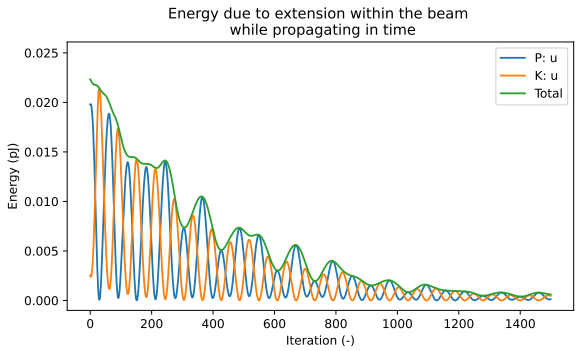
Figure 4.5: Rollout of the trained surrogate model of 4000 steps, showing the behaviour of the different degrees of freedom at different nodes.

To further investigate the behaviour of the surrogate model upon roll-out, it is interesting to look at the energy behaviour. Figure 4.6 shows the energy-behaviour split out in the kinematic, and potential energies within every separate degree of freedom using the discretization presented in Appendix B.3. It is clearly shown that only the potential energy of the lateral degree of freedom has an un-physical energy gain, while the other processes have about the same conservative energy profile as the FEM solution.

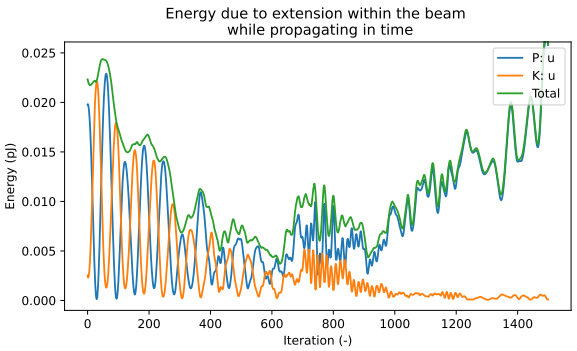
This potential energy caused by lateral displacement is solely dependent on the shear strains within the beam, as can be seen in Eq. B.2. It is therefore concluded that it is only the shear strains that grow into hugely incorrect values from the very start of the rollout. Furthermore, since the rotational behaviour stays quite consistent, it can for now be assumed that the shear- and rotational strain are fully decoupled. The increase of the shear strain has to lead to either an increase in the section rotation ϕ and/or the change of deflection dw/dx, following from Eq. 4.12[7]. Given the decoupling of γ and ϕ which was mentioned earlier, the error within the shear strain solely has an effect on the derivative of the deflection. When considering Eq. 4.13[8] the growth of the derivative of the deflection line, and thus the growth of the shear strains, can only be decoupled within the surrogate model if the entire shear term is not taken into account.

$$\frac{\partial w}{\partial x} = -\phi - \gamma \tag{4.12}$$

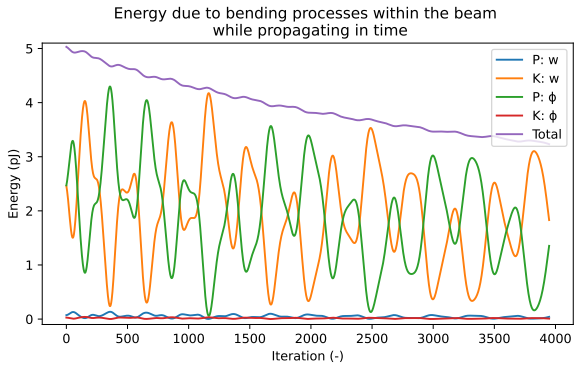
[7] Same as Eq. 4.3
[8] Same as Eq. 4.8



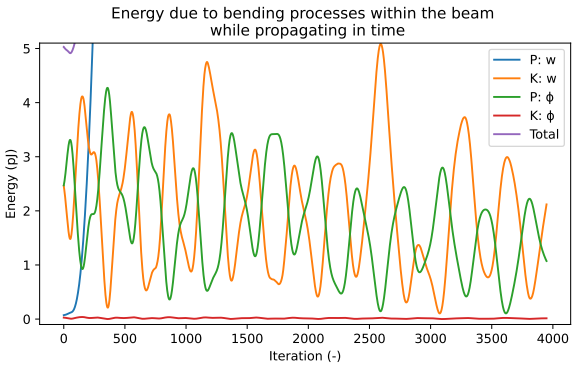
(a) Axial energy FEM



(b) Axial energy surrogate



(c) Bending-, shear-energy FEM



(d) Bending-, shear-energy surrogate

Figure 4.6: Rollout of the trained surrogate model of 4000 iterations, showing the energy behaviour of the FEM simulation and the surrogate model.

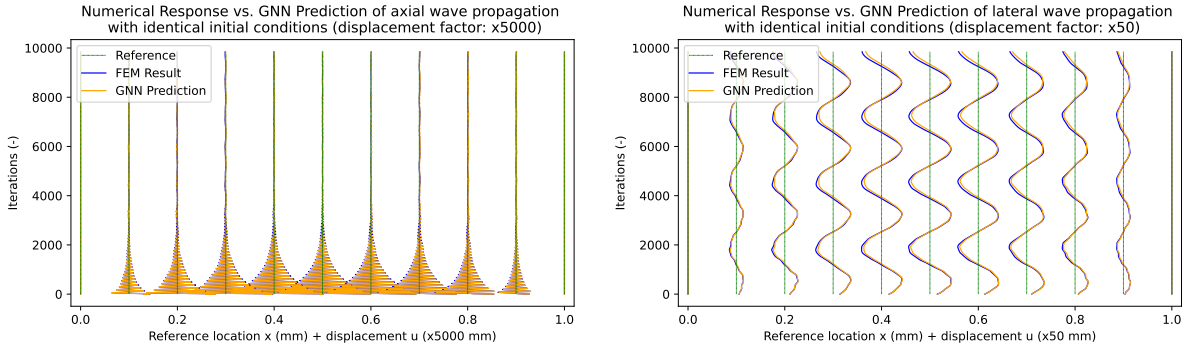
$$EI\frac{\partial^2\phi}{\partial x^2} - \rho I\frac{\partial^2\phi}{\partial t^2} - GA\left(\frac{\partial w}{\partial x} + \phi\right) = 0 \tag{4.13}$$

From this logical reasoning it can be concluded that the proposed scheme only trains on the Bernouilli bending behaviour. As a result, spurious-like behaviour is introduced on the lateral degree of freedom upon rollout. To find a solution to this issue, four ideas come to mind. Firstly, one could remove the extra degree of freedom by modelling the beam behaviour using only the information on the lateral movement of each node. Secondly, the difference between the shear- and bending-induced acceleration might be minimized over the entire dataset. This could prevent the model to overfit on either of the two processes. Thirdly, the loss function can be determined by the error multiple integration steps ahead with the intention to capture this unstable behaviour. Lastly, the shear-angle acceleration can be calculated given the surrogate outputs. This shear-angle acceleration could be used as an extra term in the loss function.

The first option, where the model is simplified by removing a degree of freedom, is worked out in the next paragraph. The second option is not feasible within the boundaries of this research since the end purpose is to model the dynamic behaviour of lattice structures in impact events. To achieve data without a large difference between the acceleration as a consequence of either bending or shear deformation, one would have to tweak the dimensions or stiffnesses of the lattice in a way that is not in line with the typical properties of these lattice structures. Within the boundaries of the practically acceptable dimensions, a second analysis was run on a beam of 0.2 mm thickness, which showed similar behaviour. The last two proposed methods are elaborated on in section 4.5 and 4.6 respectively.

4.4. Reduced Model

As presented in the previous section, the proposed solution method suffers from issues regarding an uncontrolled growth of the shear deformation. By only modelling the lateral movement of the beam, one degree of freedom is omitted[9]. This modification is proposed since it might acquire stable solutions when the shear influence is negligible. However, it should be noted that, while the shear and bending deformation can be superposed, there is no way to distinguish the two processes given only the line of deflection.
In order to fully remove the section rotations, the current-step rotation per node, the rotation-difference per node, and the rotation-derivative per edge are removed from the input features. No other changes were made to the NN architecture or training.



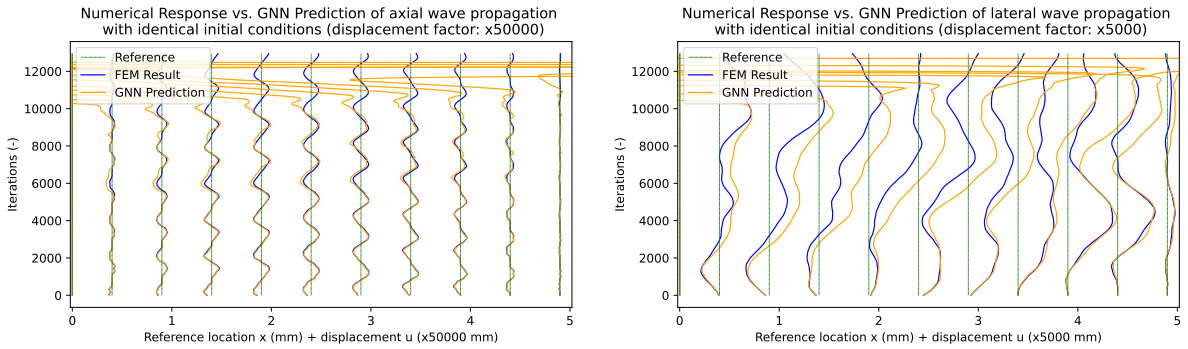
(a) Axial displacements over time

(b) Lateral displacements over time

Figure 4.7: Rollout of the trained surrogate model of 10000 steps, showing the behaviour of the different degrees of freedom at different nodes. For a beam of length 1 mm with 11 equally spaced nodes.

Figure 4.7 shows the performance upon rollout given the proposed changes by presenting the displacement of the two remaining DOFs. It immediately becomes clear that this model shows long-term stability and little deviation from the FEM solution. It can be concluded that at least for this specimen, in which the behaviour is dominated by the first eigenmode oscillation the proposed DOF-reduction gives a well-performing model.
However, when we start changing the domain by lengthening the beam, problems start occurring. The same network was trained and rolled out for a specimen which is five times longer, while the element size is kept equal[10]. The results of this test, displayed in Figure 4.8, show a clear diversion of the surrogate rollout from the FEM solution which is alike to the diversion shown in Figure 4.5. Furthermore, at around 10000 time steps the simulation becomes unstable. It is clear that for larger domains, the model with the proposed alteration does not perform well.
This inability to perform for larger domains could be explained by the increase in variation of vibration modes. As mentioned before, the Timoshenko displacement is a superposition of the deflection due to bending and the deflection due to shear. However, these two processes are modelled by different differential equations and scale differently to variations in wavelength. This means that, while it is possible to calculate the total displacement given the section forces, it is theoretically impossible to calculate the section forces given the total displacement.
Suppose that the Neural Network approximates the one-step acceleration behaviour by making a fit to the first eigenmode of the system. It will predict the behaviour of a specimen which mainly vibrates in that one eigenmode quite well as shown in Figure 4.7. However, with a five times longer beam with five times as many DOFs, five times as many modes can exist, which in turn results in a larger variation in wavelengths. Now it is no longer sufficient to fit onto one mode since the variation in vibration modes becomes significant and thus causes physically unrealistic behaviour.

[9] It could be argued that the lateral degree of freedom w should be removed since the uncontrolled growth arises in this DOF. However the rotational DOF ϕ does not give us the information desired in post-processing and is therefore more practical to omit.
[10] Leading to five times as many elements.



(a) Axial displacements over time

(b) Lateral displacements over time

Figure 4.8: Rollout of the trained surrogate model of 13000 steps, showing the behaviour of the different degrees of freedom at different nodes. For a beam with a length of 5 mm with 51 equally spaced nodes. (Only every 5th node is plotted)

4.5. Multi-step error

To fix the unstable energy behaviour another suggestion is to modify the loss function to take into account the error multiple steps ahead. This way the NN gets an incentive to simulate stable behaviour specifically. To this end, a basic multi-step procedure is implemented where the predicted accelerations get integrated into a new kinematic state on which subsequently new predictions can be performed. This process is iterated a predefined number of times to get the desired number of steps.

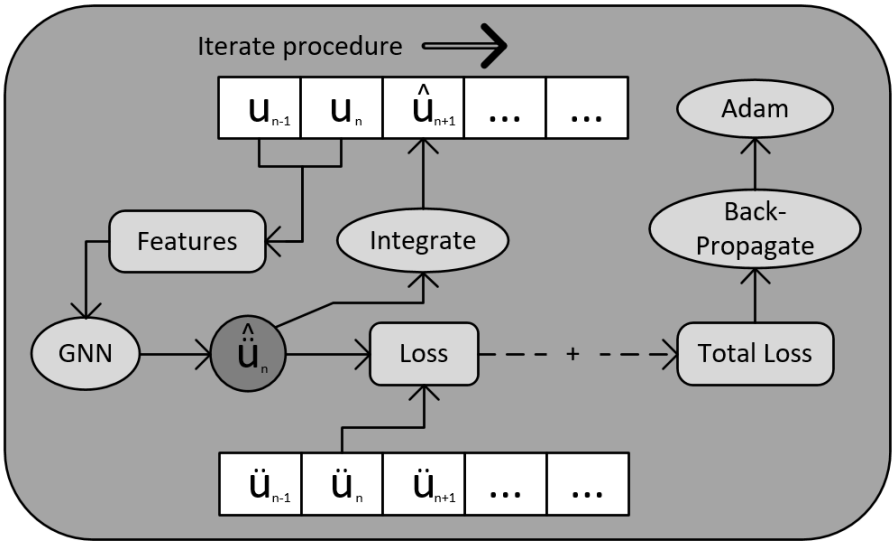


Figure 4.9: Multi-step based loss-function algorithm

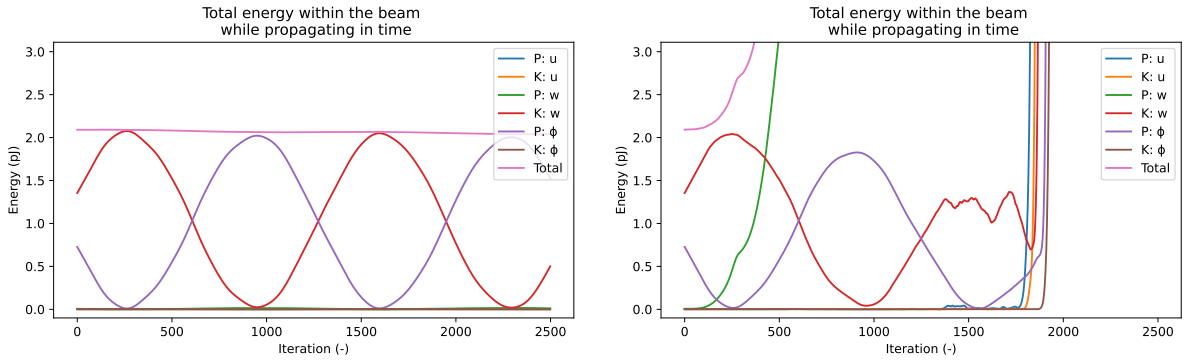
At each step, the predicted acceleration gets compared to the truth value of the original FEM simulation. The resulting MSE gets added to the total loss for every iteration as shown in Eq. 4.14, where the first step prediction gets weighted with a half and every next step gets equally weighted such that the sum of all weights μ is equal to one.

$$Loss = \sum_{n=0}^{m} \mu \cdot MSE\left(\ddot{u}_n, \hat{\ddot{u}}_n\right) \tag{4.14}$$

$$\mu = \begin{cases} 0.5, & \text{if } n = 0 \\ \frac{1}{2n}, & \text{otherwise} \end{cases} \tag{4.15}$$

The adjusted loss function with a 16-step prediction was applied to train the same network as presented before. However, no stable results could be achieved. Figure 4.10 shows the energy behaviour of a

beam upon rollout of the trained network compared to the true behaviour. Similar behaviour occurs as shown in the unmodified case where the potential shear energy shows significant unrealistic behaviour after a limited number of steps. The energy arising from the other physical processes stays close to the truth for a considerable amount of iterations after the shear energy shoots off.



(a) Total energy FEM

(b) Total energy surrogate

Figure 4.10: Rollout of the trained surrogate model of 3500 iterations, showing the energy behaviour of the FEM simulation and the surrogate model.

It can be concluded that the multi-step modification which evaluates the loss 16 steps ahead does not significantly change the performance. Looking at both Figure 4.10 and 4.6 it is clear that the significant unstable growth of the shear-potential energy sets in after around 50 iterations or later. If a multi-step loss modification would be done which looks this far ahead the erroneous behaviour could conceivably be captured better. However, every step in this procedure is rather computationally expensive since it requires a copy of the data and an integration procedure over the whole batch.
Furthermore, it is questionable to what extent the MSE of the accelerations captures this energy behaviour. The erroneous behaviour is shown in the potential energy which is not directly acceleration-dependent while the kinetic energy stays close to the truth for a long time. For this reason, it is conceivable that this energy error hardly influences acceleration behaviour. Therefore it should be considered upon further investigation to change the multi-step loss function.

4.6. Addition of Fictitious Output

In the previous sections it was found that the trained network did not capture the correct behaviour for the shear deformation which resulted in uncontrolled growth upon rollout. To counter this problem, an extra 'Fictitious' output is added, which is used to punish the network for this unphysical behaviour by adjusting the loss function. This Fictitious output is only dependent on the graph structure, input features and the output of the model ($\ddot{x}$, $\ddot{z}$ & $\ddot{\phi}$) and describes the shear-acceleration $\ddot{\gamma}$ (Fig. 4.11). It is important to note that the fictitious output is merely a means to ensure that the actual outputs are approximately consistent with relation 4.3, so there are no alterations to the network itself. By taking the second time derivative of this relation it can be stated that for it to hold true the following condition needs to be satisfied:

$$\frac{\partial^3 w}{\partial t^2 \partial s} = -\frac{\partial^2 \phi}{\partial t^2} - \frac{\partial^2 \gamma}{\partial t^2} \tag{4.16}$$

The second time derivative of ϕ is already given as an output of the network. To calculate $\partial^3 w / \partial t^2 \partial s$ the accelerations $\ddot{x}$ and $\ddot{z}$ need to be decomposed in the direction lateral to the local edge, which gives $\ddot{w}$. Thereafter the linearized acceleration of the deflection derivative can be calculated as:

$$\frac{d\ddot{w}}{ds} \approx \frac{\ddot{w}_{i+1} - \ddot{w}_i}{\Delta s} \tag{4.17}$$

Given both relations it is possible to express $\ddot{\gamma}$ using the network-outputs. By taking the Mean Squared Error of the predicted- and the true shear-angle acceleration an extra loss value is gathered. This loss value is appended to the losses of the outputs after which the mean is taken to update the network. It

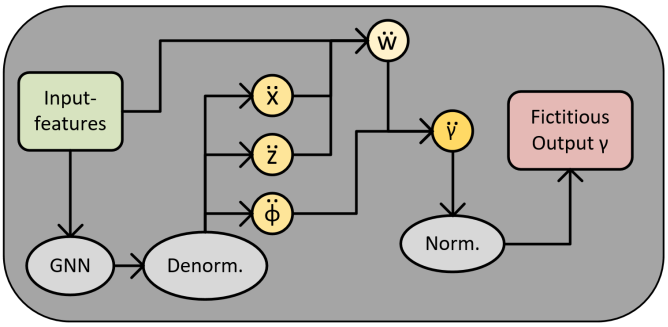


Figure 4.11: Generation of fictitious network output to constrain network to Equation 4.3

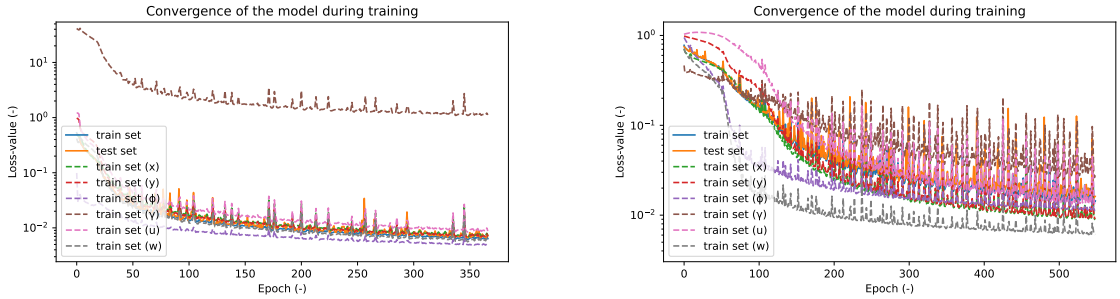
should be noted that the above expressions do not hold for the normalized values of the in- and outputs. Therefore these have to first be de-normalized to allow for the above operation. Before taking the MSE of the shear accelerations it is again normalized using the overall distribution of the shear acceleration and zero-mean unit-variance normalization. The full algorithm is shown in procedure 1.

Procedure 1 Calculation of shear-angle acceleration

Input: Vectors describing each edge $[\overline{x}, \overline{z}]$, DOF accelerations $(\ddot{x}, \ddot{z}, \ddot{\phi})$, graph $G(V, E)$ describing the beam
Denormalize all the given accelerations
Edge-length $l_e = \sqrt{\overline{x}^2 + \overline{z}^2}$
for each edge e_i in E **do**
Angle $\alpha = tan^{-1}(\overline{z}_i / \overline{x}_i)$ in the range $(0, \pi)$
Determine nodes (m, n) defined by edge e_i
Deflection acceleration $\ddot{w}_m = \ddot{z}_m cos(\alpha) - \ddot{x}_m sin(\alpha)$
Deflection acceleration $\ddot{w}_n = \ddot{z}_n cos(\alpha) - \ddot{x}_n sin(\alpha)$
Derivative-Deflection acceleration $(d\ddot{w}/ds)_i = (\ddot{w}_n - \ddot{w}_m)/l_e$
The average section-rotation acceleration $\ddot{\phi}_i = (\ddot{\phi}_n + \ddot{\phi}_m)/2$
Shear-angle acceleration $\ddot{\gamma}_i = -(d\ddot{w}/ds)_i - \ddot{\phi}_i$
end for
Normalize $\ddot{\gamma}$ by zero-mean and unit-variance normalization
Output: Normalized shear-angle accelerations $\ddot{\gamma}$

The same network as proposed earlier was trained using the modified loss-function. Figure 4.12 shows the shear-acceleration loss per epoch with and without the modification[11]. It shows that in the unmodified case the shear-loss did decrease, however, its performance is consistently two orders of magnitude worse than the average loss[12]. The reason for this behaviour lies in the separate normalization of the shear-angle acceleration. Since this acceleration is dependent on the difference between the angular acceleration of the cross-section and the line of deflection the error will be dependent on these two processes. However, the deviation of the shear acceleration is orders of magnitude lower which makes the normalized MSE orders of magnitude bigger.
In the modified case (Fig. 4.12b) it can be observed that the one-step prediction of the shear-rotation can be trained. However, the total loss converges to worse values than before. To see if this surrogate model gives us any valuable results the trained network is rolled out. It is observed in this rollout (Fig. 4.13) that the behaviour already becomes unphysical within the first 300 iterations for both the potential energy in the axial and the lateral direction.

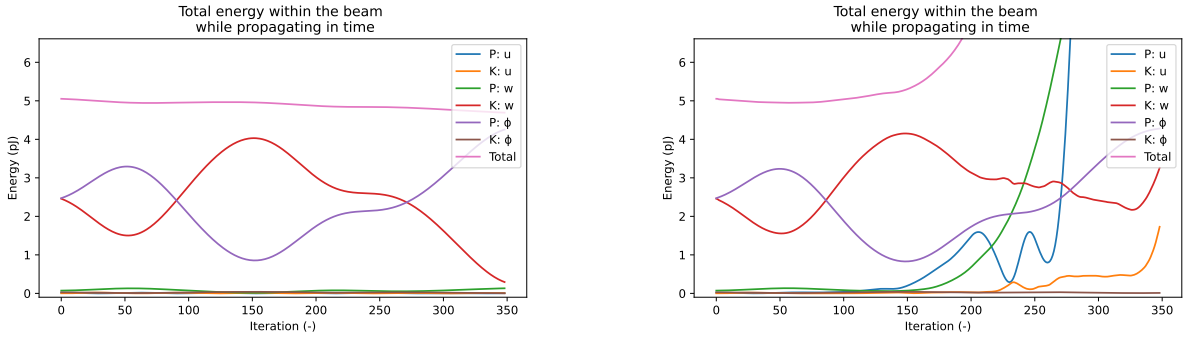
[11] In the unmodified training case the per step shear-loss was calculated but not used to update the network during training.
[12] The loss-values do not show to be starting around the same order of magnitude at the first epoch. This is due to the fact that the network already trains batches during the first epoch. Only the average of all batch losses is shown.



(a) Regular training

(b) Training with fictitious output $\ddot{\gamma}$

Figure 4.12: The same Surrogate model trained using the original scheme vs. including a fictitious shear-output with an altered hidden layer size.



(a) Total energy FEM

(b) Total energy surrogate

Figure 4.13: Rollout of 400 iterations of the trained surrogate model, with shear rotation included in the loss function. Showing the energy behaviour of the FEM simulation and the surrogate model.

A multitude of hidden-layer- and MPL sizes have been investigated. Also, different approaches were used to include the fictitious output error in the loss function. These included different weights and a gradual introduction by increasingly weighting the fictitious error within the total loss. However, none of these modifications gave a significantly different result upon rollout.
Lastly, given a similar procedure as introduced, the local axial strain could also be implemented as a fictitious output (Fig. 4.14). Any combination of the actual and fictitious output gave similar or worse results.

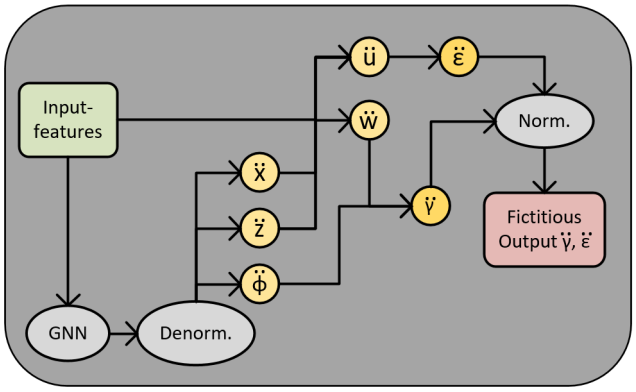


Figure 4.14: Generation of fictitious network output to constrain network to Equation 4.3 and correct global-local extension decomposition.

5

Discussion

Within this research it was found that the complexity of lattice structures could not be modelled in a way which achieved a reliable, generalizable and stable outcome using a time-stepper method with an architecture which is alike to that of Pfaff et al., 2020. However, considering the empirical approach taken, this does not conclude that the simulation of waves through a lattice structure is impossible.
What does stand out is the difference in mathematical complexity between the modelling of lattice structures and the simulations already performed by a similar architecture like in Chapter 3 and in several studies (Sanchez-Gonzalez et al., 2020 & Pfaff et al., 2020). Apart from the fact that a coupled set of 2^{nd}-order PDEs needs to be modelled, the dynamics itself is governed by 3 different types of physical behaviour which all have a different range of natural frequencies.

It is interesting to discuss the composed research sub-questions for possible further research. These are indicative of the general use of GNN time-stepper models to simulate dynamic behaviour. After discussing the sub-questions, recommendations based on research experience are presented. Lastly, recommendations are made for further research.

What architectural choices are relevant when modelling dynamics using Graph Neural Network time-stepper models?
Firstly, throughout this research it was found that the inclusion of all physical quantities within the FEM simulation is needed to find a well-functioning surrogate model. The exclusion of information which is used to gather the training data can lead to erroneous or unstable behaviour as shown in section 4.4. This does not go for physical quantities which are kept constant throughout the data set.
Within the architecture of the GNN itself the number of MPLs is of the most importance. As was shown in section 3.4.7, a single-MPL-based model functioned significantly worse for the surrogate trained on data generated using an implicit integration scheme. While this decrease in performance can partly be ascribed to the reduction of learnable weights, it can be deduced from section 3.4.6 that a double-MPL-based model with half the units per layer still performed considerably better[1]. Therefore it can be concluded that a significant part of the performance change is not due to the decrease in learnable weights but to the reduction in MPLs.
Given the time limit and resources it could not be determined whether the application of noise according to the proposed scheme in Appendix A.2 had any favourable influence on the rollout error. It can be concluded however that the effect of the inclusion of the proposed scheme does not majorly influence the performance at inference.

[1] Half the units per layer leads to only a quarter of the learnable weights.

How well can a GNN time-stepper model generalise to different configurations in space?
As for the continuum case presented in Chapter 3 it is clear that the surrogate model trained on a small domain can be rolled out on larger domains. Furthermore, the model shows the capability to perform well on in-homogeneous meshes, allowing for large flexibility upon inference of the surrogate model.
The Timoshenko-beam surrogate model using a scheme with a reduced DOF, as elaborated in section 4.4, showed physically realistic behaviour with considerable long-term stable behaviour for a small beam size with few vibration modes. Upon changing the beam length, which introduced more vibration modes, the number of stable iterations decreased. For the full Timoshenko model no stable roll-outs were found, therefore no conclusions can be drawn about whether this model generalizes to different domains.

Recommendations based on research experience
Several models were investigated which are mentioned but not directly displayed in this report. For the convenience of the reader, some experiences that arose from these experiments are listed below.

- Acceleration-based time-stepper models performed considerably better than velocity-based integrations or direct displacement predictions with the same GNN architecture. It was generally observed that the use of direct displacement predictions leads to either stationary or unstable rollouts. Velocity integrations showed nonphysical inertial behaviour and were unstable.
- The inclusion of dataset and batch-normalisation is sometimes vital to get a converging loss function. Many models tended to predict an all-zero outcome for every node after a limited amount of epochs. After which no further decrease in loss could be observed. Adding the normalizations mostly helped to overcome this issue.
- Models with multiple node- and edge updates (multi MPL) performed significantly better when the learnable weights were not shared between layers.

Further research
As far as the results obtained are concerned, several things could be studied while keeping the same model architecture. Firstly, the models could be trained an order of magnitude longer. This could result in better performance than observed in this report, as some of the convergence plots still show decreasing behaviour when training is completed. Secondly, there is a discrepancy between the FEM integration scheme used to generate the data for the continuum case and the Timoshenko case. For a fair comparison between the two, it would be valuable to generate the Timoshenko data set using a backwards-Euler scheme.
With regard to the inability to model the behaviour of Timoshenko beams, various modifications were proposed and tested. Although they failed to achieve stable solutions individually, the combination of any of the solutions could be more successful. The most promising would be a combination of the multi-step loss function taking into account the kinematic relations using a fictitious output. The other solutions intrinsically hamper the generalizability of the surrogate and are therefore less interesting to research further.
Looking at the big picture, the bottleneck found was the inability to model the set of coupled second-order PDEs which define the behaviour of a Timoshenko beam. If, in the future, a surrogate model is found that can model a similarly complex set of PDEs, it could be a good starting point from which to create a GNN architecture simulating dynamic lattice behaviour.

6

Conclusion

The aim of this research was to investigate whether the application of GNN time-stepper models could replace existing FEM software in the simulation of the dynamic behaviour of lattice structures. Based on the results shown in chapter 3 and various literature, a GNN can be used to approximate a single second-order PDE. However, for the modelling of lattice structures, it is necessary to model both shear and bending behaviour in a coupled set of second-order PDEs. Based on the results shown in chapter 4, no successful surrogate model could be found to simulate this more complex behaviour. Furthermore, no literature could be found on NN models performing this task.
The main difficulty encountered was the modelling of vibration modes with varying wavelengths while maintaining a stable simulation. For each model which was evaluated, instabilities arose in one of the three DOFs defined by the coupled PDEs. The construction of different loss functions to correct this behaviour did not improve stability. Furthermore, it was found that excluding the unstable DOF is not a feasible solution as this is fundamental information needed to model the behaviour of the beam.
Although, given the empirical nature of this research, it cannot be ruled out that GNNs can be applied to simulate the dynamic behaviour of Timoshenko beams, it is shown that modelling these more complicated physical processes is more challenging than modelling regular continuous mechanics. The same architectural strategies were applied to the regular and coupled second-order PDE cases. While the regular case was stable for all tested inputs in the training range, the coupled second-order PDE simulation led to unstable results for any input.

A

Miscellaneous Information Continuum Dynamics

A.1. Data set

The train- and validation set for the 1-Dimensional case consists of a set of p specimens with each a unique mesh and set of initial conditions. To ensure reproducible results the initial conditions are created at random by adding a uniform random initial displacement to a uniform randomly chosen node. This addition is done two times for all the specimens used in this report. For specimen generation see Procedure 2.

Procedure 2 Creation of specimen (1D bar)

Input: Length $'l'$, number of nodes $'n'$, magnitude of element-size deviation $'\kappa'$
  Generate uniform mesh x^u given the length and number of nodes.
  $dx = l/(n-1)$
  for each element in x^u **do**
    Draw random number $'r'$ in the range $(-1, 1)$
    Perturb node x_k^{n-u} $+= \kappa \cdot r \cdot dx$
  end for
  Generate a zero-vector $'u_0'$ of of size $'n'$
  for predefined number of times **do**
    Draw random integer $'n_r'$ in the range $(1, n-1)$
    Draw random number $'r'$ in the range $(-0.1, 0.1)$
    $u_{0;n_r}$ $+= r * dx$
  end for
  for number of iterations $'i'$ **do**
    Calculate next state u^{i+1} using the proposed FEM scheme
    Calculate the current acceleration
    Apply noise and calculate acceleration based on central difference $a^i = \left(u^{i-1} - 2u^i + u^{i+1}\right)/\Delta t^2$
  end for
  Remove the first $'m'$ time-steps from u and a
Output: One sample rolled out for $'i'$ time-steps

The total set of specimens is collected to determine and apply the normalisation. Thereafter, for each time step of each specimen, the input features and truth value is determined and appended to the total data set of samples to be trained on. This total data set is shuffled and divided into a train- and validation set with a respective ratio of 80/20.

A.2. Noise

For the generation of noise, the scheme from Sanchez-Gonzalez et al., 2020 is used. Here for every data sample the current displacement field is perturbed and the data of the derivative is corrected to be consistent. Since the acceleration is a second derivative, and the historical data of the displacement field is taken as an input, there can not be a consistent true correction, as remarked by Pfaff et al., 2020. Therefore the proposed correction is applied, where γ is a hyper-parameter, $\tilde{\ddot{x}}_i^P$ is the correct acceleration based on the perturbed displacement, and $\tilde{\ddot{x}}_i^V$ is the correct acceleration based on the perturbed velocity.

$$\tilde{\ddot{x}} = \gamma \tilde{\ddot{x}}^P + (1-\gamma)\,\tilde{\ddot{x}}^V \tag{A.1}$$

Procedure 3 Application of Noise

Input: Displacement-fields for entire run $'u'$, hyper-parameter $'\gamma'$, hyper-parameter $'\epsilon'$, time-increment Δt
Determine the scaled noise-value $\epsilon^* = \epsilon \cdot \sigma(u)$
for each time-step in u **do**
Draw a random number $'r'$ from a normal distribution with mean 0 and standard-deviation ϵ^*
$\tilde{u}^i = u^i + r - \left(u^{i-1} - \tilde{u}^{i-1}\right)$
end for
Save perturbed displacement-field separately
for each time-step in u **do**
Calculate the velocity based acceleration $\tilde{\ddot{u}}^{V;i} = \left(u^{i-1} - u^i - \tilde{u}^i + u^{i+1}\right)/\Delta t^2$
Calculate the displacement-based acceleration $\tilde{\ddot{u}}^{P;i} = \left(u^{i-1} - 2\tilde{u}^i + u^{i+1}\right)/\Delta t^2$
$\tilde{\ddot{u}}^i = \gamma \tilde{\ddot{u}}^{P;i} + (1-\gamma)\,\tilde{\ddot{u}}^{V;i}$
end for
Output: Perturbed displacement- and acceleration-field

An important note is that within the input features the perturbed displacement field of the evaluated time step is used, and the non-perturbed displacement field for the preceding time step.

A.3. Physical, discretisation, NN and data properties

For the base-dataset, which is used unless specified otherwise, the physical properties of the specimens can be found in Table A.1 and the hyper-parameters in Table A.2.

Property	Value	Unit
E	$2.1 \cdot 10^{11}$	
$N/m^2\ A$	$1 \cdot 10^{-2}$	m
ρ	$7.8 \cdot 10^{3}$	kg/m^3
L	1.0	m
Δt	10^{-6}	s
n	50	-

Table A.1: Physical and discretisation quantities used for the specimens within the train- and validation set

Property	Value
iterations per specimen (i)	1000
num. of samples (p)	200
Non-uniformity (κ)	0.2
Noise parameter (γ)	0
Excluded steps (m)	60
Layer-size (s)	64
Message Passing Layers (MPL)	2

Table A.2: Hyperparameters for the data set

A.4. Discretized calculation of energy

For the one-dimensional case, comparisons are made in the energy behaviour between different models. To perform these analyses it is needed to approximate the energy based on the kinematic state of the discretized continuum. This is done by assuming the bar to be a mass-spring system, where the nodes represent the masses, and the edges represent springs. The mass assigned to a node is calculated according to Eq. A.2 where $'x'$ represents the vector containing the initial node-positions. The discretized mass of the nodes at both ends does not have to be calculated.

$$m_i = \rho \cdot A \cdot \frac{x_{i+1} - x_{i-1}}{2} \tag{A.2}$$

The spring constant of the edges is determined by Eq. A.3 where L_i is the length of the edge.

$$k_i = \frac{EA}{L_i} \tag{A.3}$$

Within the system there is a combination of kinetic and potential energy. The total energy is the sum of both as in Eq. A.4. It should be remarked that this is a quite basic approximation, and thus only valuable for means of comparison within the context of this report.

$$\mathcal{E}^T = \mathcal{E}^K + \mathcal{E}^P = \sum_{k=2}^{n-1} \frac{m_k \cdot \dot{u}_k^2}{2} + \sum_{p=1}^{n-1} \frac{k_p \cdot \left(u_{p+1} - u_p\right)^2}{2} \tag{A.4}$$

A.5. Derivation eigen-mode

For an extensional bar the partial differential equation which describes the free motion[1] is defined as Eq. A.5 (Spijkers et al., 2006). Inserting the assumed solution (Eq. A.6) with separation of variables Eq. A.7 is acquired. Simplification leads to the full separation of the spatial part of the equation (Eq. A.8). For which Eq. A.10 defines a possible solution.

$$EA\frac{\partial^2 u\,(x,t)}{\partial x^2} - \rho A\frac{\partial^2 u\,(x,t)}{\partial t^2} = 0 \tag{A.5}$$

$$u\,(x,t) = u\,(x)\sin\,(\omega t + \phi) \tag{A.6}$$

$$EA\frac{\partial^2 u\,(x)}{\partial x^2}\sin\,(\omega t + \phi) + \omega^2 \rho A \cdot u\,(x)\sin\,(\omega t + \phi) = 0 \tag{A.7}$$

$$\frac{\partial^2 u\,(x)}{\partial x^2} + \alpha^2 u\,(x) = 0 \tag{A.8}$$

$$\alpha^2 = \frac{\omega^2 \rho}{E} \tag{A.9}$$

$$u\,(x) = C_1 sin\,(\alpha x) + C_2 cos\,(\alpha x) \tag{A.10}$$

The boundary conditions for the analysed problem is a zero displacement at both sides. Therefore C_1 and C_2 can be defined to be:

$$C_1 sin\,(\alpha L_{beam}) = 0 \tag{A.11}$$

$$C_2 = 0 \tag{A.12}$$

In order to get a non-trivial solution C_1 needs to be unequal to zero. Therefore α can be defined as in Eq. A.13 for any value of n. Which results in angular frequencies as stated in Eq. A.14, where $'n'$ indicates the degree of the eigenmode. Within this report only the first and second eigenmode are relevant.

$$\alpha = \frac{n\pi}{L_{beam}} \tag{A.13}$$

$$\omega = \frac{n\pi}{L_{beam}}\sqrt{\frac{E}{\rho}} \tag{A.14}$$

[1] Free vibration, thus homogeneous PDE.

A.6. Derivation implicit acceleration 7 node test

Within the 7-node test, the acceleration is predicted by the GNN model on a zero-velocity displaced field. The true acceleration[2] in such case is:

$$\ddot{u}_{approx} = -\frac{2 \cdot u \cdot E}{\rho \cdot l_{el}^2} \tag{A.15}$$

This acceleration is not equal to the truth value in the data set because due to the FEM scheme this acceleration is calculated by an implicit relation. Eq. A.16 shows the basis of the implicit formulation for the displacement update of a single point using central difference. Which upon rewriting gives Eq. A.17. By taking into account the fact that the point has zero velocity simplification leads to Eq. A.18.

$$m\frac{u^{i-1} - 2u^i + u^{i+1}}{\Delta t^2} = ku^{i+1} \tag{A.16}$$

$$u^{i+1} = \frac{m\left(u^{i-1} - 2u^i\right)}{k\Delta t^2 - m} \tag{A.17}$$

$$u^{i+1} = \frac{-mu^i}{k\Delta t^2 - m} \tag{A.18}$$

By approximating the mass as the element length $m = \rho A l_{el}$, and the stiffness as two times the stiffness of the adjacent springs $k = 2EA/l_{el}$ rewriting leads to Eq. A.19.

$$u^{i+1} = \frac{-\rho u^i}{2\frac{E}{l_{el}^2}\Delta t^2 - \rho} \tag{A.19}$$

Substituting in the central difference formulation for acceleration gives the final form (Eq. A.20). Note here, that the 'discretized' acceleration is dependent on delta-time and is by no means the 'actual' acceleration at the evaluated time.

$$\ddot{u}_{approx} = -u\frac{\rho}{2\frac{E\Delta t^4}{l_{el}^2} - \rho\Delta t^2} - \frac{u}{\Delta t^2} \tag{A.20}$$

[2]Given the conditions as defined in section 3.4.3.

B

Miscellaneous Information Beam Dynamics

B.1. Data set

The train- and validation set for the topologically 1-dimensional Timoshenko surrogate model consists of a set of p specimens with each a unique set of initial conditions. To ensure reproducible results the initial conditions are created at random by adding a uniform random initial displacement to all nodes in a randomly generated range. This addition is done two times for all the specimens used in this report. For specimen generation see Procedure 4.

Procedure 4 Creation of specimen (Timoshenko beam)

Input: Length l, number of nodes n and maximum excitation d_{max}
  Generate uniform mesh x given the length and number of nodes.
  Element-length $dx = l/(n-1)$
  Generate zero-vectors u^0 and w^0 of of size n
  for 2 times **do**
    Draw a random floating point number r in the range $(0.1 * l, 0.7 * l)$
    Draw two random floating point numbers d_u and d_w in the range $(-d_{max}, d_{max})$
    for for all nodes i within the range $(r, r + 2 * dx)$ **do**
      $u_i^0 += d_u$
      $w_i^0 += d_w$
    end for
  end for
  Rotate the entire beam around its origin with a uniform random angle in the range $(0, \pi)$
  for number of iterations $'i'$ **do**
    Calculate next state u^{i+1} using a FEM scheme
  end for
  Remove the first $'m'$ time-steps from u and a
Output: One sample rolled out for $'i'$ time-steps

The total set of specimens is collected to determine and apply the normalisation. Thereafter, for each time step of each specimen, the input features and truth values are determined and appended to the total data set of samples to be trained on. This total data set is shuffled and divided into a train- and validation set with a respective ratio of 80/20.

B.2. Physical, discretization, NN and data properties

For the base data set, which is used unless specified otherwise, the physical properties of the specimens can be found in Table B.1 and the hyper-parameters in Table B.2.

Property	Value	Unit
E	$2.1 \cdot 10^{8}$	N/m^2
ν	0.3	-
A	$2.5 \cdot 10^{-3}$	mm^2
I	$5.21 \cdot 10^{-7}$	mm^4
ρ	$7.85 \cdot 10^{3}$	kg/m^3
L	1.0	mm
Δt	10^{-7}	s
n	11	-

Table B.1: Physical and discretization quantities used for the specimens within the train- and validation set

Property	Value
iterations per specimen (i)	10000
num. of samples (p)	40
Excluded steps (m)	60
Layer-size (s)	128
Message Passing Layers (MPL)	2

Table B.2: Hyperparameters for the data set

B.3. Discretized calculation of energy

For the evaluation of the beam surrogate models' performance, energy comparisons are made. To perform these analyses it is needed to approximate the energy based on the kinematic state of the discretized beam. The same principles are followed as described in Appendix A.4, with the addition of the need to calculate the shear and rotatory energy. If we follow the derivation by Timoshenko as elaborated in section 4.1 inertia is given as ρI integrated over the length. By taking the mass to be equal to $\rho I \Delta x$ the rotatory inertia around a node can be approximated[1]. This might seem counter-intuitive since it is not the full inertia of the beam segment is used, which would be equal to $\rho I \Delta x^3$. However, within the Timoshenko beam formulation the rotation ϕ describes the rotation of the cross-section which does not have to be equal to the rotation of the axis of deflection. The spring stiffness between two discretized rotated sections is equal to $EI/\Delta x$ which holds true if Δx is sufficiently small[2]. Given the relations described above the total energy as a consequence of rotatory inertia in the beam can be formulated as in Eq. B.1 where n describes the number of nodes in the beam. For the nodes at both ends of the beam, only half the inertia is taken since they are only connected to an element on one side.

$$\mathcal{E}_R = \mathcal{E}_R^K + \mathcal{E}_R^P = \frac{\rho I \Delta x \left(\dot{\phi}_1^2 + \dot{\phi}_n^2\right)}{4} + \sum_{k=2}^{n-1} \frac{\rho I \Delta x \cdot \dot{\phi}_k^2}{2} + \sum_{p=1}^{n-1} \frac{EI \cdot \left(\phi_{p+1} - \phi_p\right)^2}{2\Delta x} \tag{B.1}$$

Secondly, the energy as a consequence of the lateral displacement of the sections needs to be found. It would be incorrect to take the relative vertical displacement of two nodes as the shear strain since with this approximation the vertical displacement as a consequence of bending interaction is also taken into account. To correct for this the shear strain γ is acquired using relation 4.3. Where the derivative of the axis of deflection can be approximated as the difference between the nodes on either side of the 'shear-spring', and the average of both angles ϕ is taken. The kinematic energy of the lateral movement is acquired by assuming the nodes to be point masses using the same approximation as for extension.

[1] Given that the mesh is homogeneous, so all elements have the same length, and that the rotation is equal for every slice in the same discretized part.

[2] This can easily be derived from the kinematic relation $M = \phi' EI$.

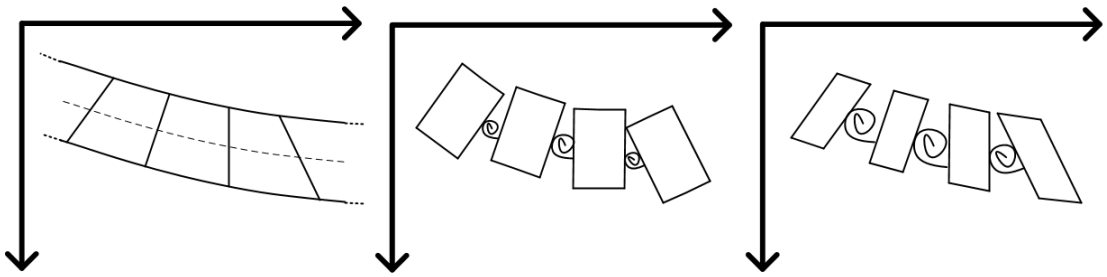


Figure B.1: Discretizing rotatory inertia. From left to right: Beam-segment, Inertia equal to $\rho I \Delta x^3$ (wrong) and Inertia equal to $\rho I \Delta x$ (Adapted definition)

$$\mathcal{E}_L = \mathcal{E}_L^K + \mathcal{E}_L^P = \sum_{k=2}^{n-1} \frac{\rho A\, \Delta x \cdot \dot{w}_k^2}{2} + \sum_{p=1}^{n-1} \frac{GA\Delta x \cdot \left(-\frac{\phi_{p+1}+\phi_p}{2} - \frac{w_{p+1}-w_p}{\Delta x}\right)^2}{2} \tag{B.2}$$

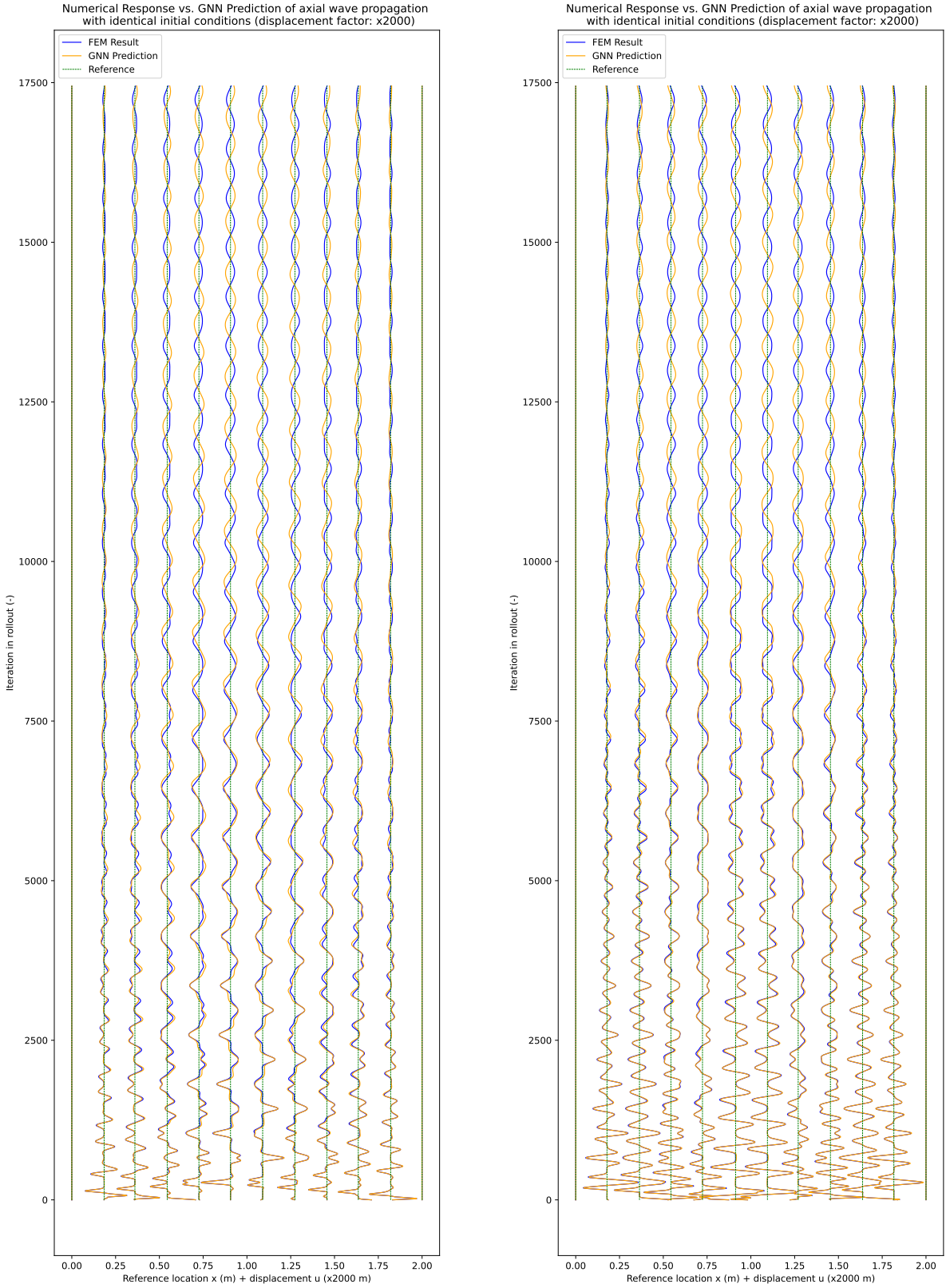
Applying the same Energy-approximation for the extension as presented in A.4, the total energy within the system can be approximated by Eq. B.3.

$$\mathcal{E}^T = \mathcal{E}_R + \mathcal{E}_L + \mathcal{E}_A \tag{B.3}$$

C

Appendix Results Continuum Model

C.1. Long rollout default model



(a) (b)

Figure C.1: Behaviour of the default GNN upon rollout for 17500 steps.

Bibliography

Bashforth, F., & Adams, J. C. (1883). *An attempt to test the theories of capillary action by comparing the theoretical and measured forms of drops of fluid: With an explanation of the method of integration employed in constructing the tables which give the theoretical forms of such drops*. Cambridge [Eng.] University Press.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., … Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *CoRR*, *abs/1806.01261*. http://arxiv.org/abs/1806.01261

Demirdag, O., & Murat, Y. (2009). Free vibration analysis of elastically supported timoshenko columns with attached masses using fuzzy neural network. *Journal of Scientific Industrial Research*, *68*, 285–291.

(DHPC), D. H. P. C. C. (2022). DelftBlue Supercomputer (Phase 1).

Eugster, S. R. (2015). Springer Cham. https://doi.org/10.1007/978-3-319-16495-3

Euler, L. (1744). Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes, sive solutio problematis isoperimetrici lattissimo sensu accepti. *65*. https://scholarlycommons.pacific.edu/cgi/viewcontent.cgi?article=1064&context=euler-works

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. *CoRR*, *abs/1903.02428*. http://arxiv.org/abs/1903.02428

Francisco, M. B., Pereira, J. L. J., Oliver, G. A., da Silva, L. R. R., Jr, S. S. C., & Gomes, G. F. (2021). A review on the energy absorption response and structural applications of auxetic structures. *Mechanics of Advanced Materials and Structures*, *0*(0), 1–20. https://doi.org/10.1080/15376494.2021.1966143

Freund, J., & Karakoc, A. (2016). Shear and torsion correction factors of timoshenko beam model for generic cross sections. *Research on Engineering Structures Materials*, *2*. https://doi.org/10.17515/resm2015.19me0827

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *14*(3), 1–159.

Hartsuijker, C., & Welleman, J. W. (2016). *Engineering mechanics: Stresses, strains, displacements* (Vol. 2). Springer.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. https://doi.org/10.48550/ARXIV.1502.03167

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization.

Koi, D. (2022). *Connected lines and dots*. https://unsplash.com/photos/GU_nNLVna_4

Legaard, C. M., Schranz, T., Schweiger, G., Drgoňa, J., Falay, B., Gomes, C., Iosifidis, A., Abkar, M., & Larsen, P. G. (2021). Constructing neural network-based models for simulating dynamical systems. https://doi.org/10.48550/ARXIV.2111.01495

Meister, F., Passerini, T., Mihalef, V., Tuysuzoglu, A., Maier, A., & Mansi, T. (2020). Deep learning acceleration of total lagrangian explicit dynamics for soft tissue mechanics. *Computer Methods in Applied Mechanics and Engineering*, *358*, 112628. https://doi.org/https://doi.org/10.1016/j.cma.2019.112628

Papadopoulos, V., Soimiris, G., Giovanis, D., & Papadrakakis, M. (2018). A neural network-based surrogate model for carbon nanotubes with geometric nonlinearities. *Computer Methods in Applied Mechanics and Engineering*, *328*, 411–430. https://doi.org/https://doi.org/10.1016/j.cma.2017.09.010

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (pp. 8024–8035).

Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., & Battaglia, P. W. (2020). Learning mesh-based simulation with graph networks. https://doi.org/10.48550/ARXIV.2010.03409

Rajasekaran, S., Khaniki, H. B., & Ghayesh, M. H. (2022). On the mechanics of shear deformable micro beams under thermo-mechanical loads using finite element analysis and deep learning neural network. *Mechanics Based Design of Structures and Machines*, *0*(0), 1–45. https://doi.org/10.1080/15397734.2022.2047721

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. W. (2020). Learning to simulate complex physics with graph networks. https://doi.org/10.48550/ARXIV.2002.09405

Spijkers, J. M. J., Vrouwenvelder, A. W. C. M., & Klaver, E. C. (2006). Dynamics of structures, part 1: Vibration of structures.

Timoshenko, S. P. (1921). Lxvi. on the correction for shear of the differential equation for transverse vibrations of prismatic bars. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *41*(245), 744–746. https://doi.org/10.1080/14786442108636264

Timoshenko, S. P. (1953). Dover Publications. https://app.knovel.com/hotlink/toc/id:kpHSMWBAH3/history-strength-materials/history-strength-materials

Wells, G. N. (2020). The finite element method: An introduction.

Xue, T., Adriaenssens, S., & Mao, S. (2022). Learning the nonlinear dynamics of soft mechanical metamaterials with graph networks. https://doi.org/10.48550/ARXIV.2202.13775

Yagawa, G., & Oishi, A. (2021). *Computational mechanics with neural networks*. https://doi.org/10.1007/978-3-030-66111-3

Yõldõrõm, B. (2014). Comparing exact and generalized regression neural network solutions for free vibration of elastically supported timoshenko beams with attached masses.

Ziemiański, L. (2003). Hybrid neural network/finite element modelling of wave propagation in infinite domains [K.J Bathe 60th Anniversary Issue]. *Computers Structures*, *81*(8), 1099–1109. https://doi.org/https://doi.org/10.1016/S0045-7949(03)00007-5