Preprint ISSN 2198-5855

Topology optimisation under uncertainties with neural networks

Martin Eigel[1] Marvin Haase[2], Johannes Neumann[3]

submitted: December 15, 2022

[1] Weierstrass Institute
Weierstrass Institute
Mohrenstr. 39
10117 Berlin
Germany
E-Mail: martin.eigel@wias-berlin.de

[2] marvinhaase@hotmail.com

[3] Rafinex Ltd.
Great Haseley OX44 7JQ
UK
E-Mail: johannes.neumann@rafinex.com

No. 2982
Berlin 2022

2020 *Mathematics Subject Classification.* 35R60, 47B80, 60H35, 65C20, 65N22, 65J10.

Key words and phrases. Topology optimisation, deep neural networks, model uncertainties, random fields, convolutional neural networks, recurrent neural networks.

The first author acknowledges partial funding by the DFG SPPs 1886 and 2298.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Topology optimisation under uncertainties with neural networks

Martin Eigel Marvin Haase, Johannes Neumann

Abstract

Topology optimisation is a mathematical approach relevant to different engineering problems where the distribution of material in a defined domain is distributed in some optimal way, subject to a predefined cost function representing desired (e.g., mechanical) properties and constraints. The computation of such an optimal distribution depends on the numerical solution of some physical model (in our case linear elasticity) and robustness is achieved by introducing uncertainties into the model data, namely the forces acting on the structure and variations of the material stiffness, rendering the task high-dimensional and computationally expensive. To alleviate this computational burden, we develop two neural network architectures (NN) that are capable of predicting the gradient step of the optimisation procedure. Since state-of-the-art methods use adaptive mesh refinement, the neural networks are designed to use a sufficiently fine reference mesh such that only one training phase of the neural network suffices. As a first architecture, a convolutional neural network is adapted to the task. To include sequential information of the optimisation process, a recurrent neural network is constructed as a second architecture. A common 2D bridge benchmark is used to illustrate the performance of the proposed architectures. It is observed that the NN prediction of the gradient step clearly outperforms the classical optimisation method, in particular since larger iteration steps become viable.

1 Introduction

Structural topology optimisation is the (engineering oriented) process of designing a construction part using optimisation algorithms under certain constraints. The resulting designs usually have a large influence on the subsequent production costs. The starting point of the process is a design domain that represents the maximum space available for the optimised component to be developed. The outcome is an information about which parts of the design space are occupied by material and which are void. Often, the task is motivated by mechanical requirements, e.g., sufficient stiffness of the constructed part with respect to assumed forces acting on it while certain predetermined points or surfaces should connect to other parts. A typical physical model for this comes from linear elasticity, describing the displacement field given material properties and forces. For the mathematical optimisation, it is repeatedly necessary to compute the stress distribution determined by the physical model in the design domain (more precisely, in the parts of the domain with material). This potentially complex computational task usually relies on the finite element method (FEM), which is based on a discretisation of the domain into elements. Most commonly, the domain is represented as mesh consisting of disjoint simplices, i.e., triangles in 2D and tetrahedra in 3D.

Since the optimisation process easily requires several hundred evaluations of the state equation to evolve the material distribution, it is of significant interest to develop techniques that reduce this computational burden. This even becomes much more pronounced when uncertainties of the model data should be considered in the computations. The treatment of uncertainties has been developed extensively from a theoretical and practical point of view in the last decade in the area of Uncertainty

DOI 10.20347/WIAS.PREPRINT.2982

Quantification (UQ). A common way to describe uncertainties is by means of random fields, whose (Karhunen-Loève) expansions depend on a possibly very large number of random variables. The parameter space spanned by these random variables leads to very high-dimensional state problems for which derived optimisation problems are very difficult to solve.

This paper investigates the application of a trend in scientific computing for current topology optimisation methods, namely the use of modern machine learning techniques. More precisely, our objective is to improve the efficiency of the structural topology optimisation problem by predicting gradient steps based on generated training data. This efficiency gain directly transfers to our ability to compute much more involved risk-averse stochastic topology optimisation problems with random data. In this case, the topology is optimised with an adjusted cost functional including the CVaR (conditional value at risk), by which unlikely events can be taken into account in contrast to just optimising with the mean value of possible load scenarios. In addition to random loads, we also include random material properties which, e.g., can enter the model in terms of material errors or impurities. We emphasise that risk-averse optimisation based on some risk measure is a timely topic, which plays a role in many application areas. Despite its relevance, this type of problem has not been covered widely in the literature yet. In fact, the authors are not aware of any other machine learning assisted approach for risk-averse topology optimisation. This might be due to the more involved mathematical framework and the substantially higher computational complexity. To achieve performance benefits with topology optimisation in this paper, we adapt concepts from the field of deep learning to approximate multiple iterations of of the optimisation process and make the overall optimisation more efficient.

The goal of topology optimisation is to satisfy the technical requirements of a component (for instance stiffness with respect to certain loading scenarios) with minimal use of material. There are different approaches to describe the topology in a flexible way such that substantial changes are possible. We follow our previous work in [10] and use a phase field model which describes the density of material with a value in $[0, 1]$. The starting point is the definition of a physical design space available for the component under consideration. This space is completely filled with a material in the sense of an initial solution. Furthermore, all points at which loads act on the component, as well as the type of the respective load, are prescribed. The optimisation aims to achieve a homogeneous, minimum possible deformation at all optimised points of the component under the imposed (possibly continuous and thus infinitely many) loading scenarios. Here, a minimum compliance corresponds to a maximum stiffness. In general, even solving the underlying partial differential equation (PDE) of this problem for deterministic coefficients of the PDE is already a complex task. Furthermore, PDE coefficients which describe material and the loads have a strong influence on the resulting topology, i.e., even small changes in these coefficients can lead to large differences in the resulting topology. This results in considerable computational effort in the stochastic settings, since the solution has to be calculated for sufficiently many data realisations to become reliable. Hence, the modelling of these stochastic settings for example (with the most obvious approach) by a Monte Carlo (MC) simulation increases the required iteration steps linearly in the number of simulations.

A method to numerically tackle topology optimisation uncertainty was presented in [9]. In this paper we extend the previous work by introducing Deep Neuronal Networks (DNN) that are designed to provide a prediction of the next gradient step. Since topologies discretised with finite elements can be represented as images, Convolutional Neural Networks (CNN) seem natural candidate architectures for this task and there has already been some research on this approach for the deterministic setting. An introduction is presented in [20] where the conventional topology optimisation algorithms is replicated in a computationally inexpensive way. Furthermore, a CNN is used in [4] to approximate the last iteration steps of a gradient method of a topology optimisation after a fixed number of steps to refine a “fuzzy” solution. A CNN architecture is also used in [26] to solve a topology optimisation problem and trained with large amounts of data. The resulting NNs were able to solve problems with boundary conditions different to their training data. In [23], the problem is stated as an image segmentation task and an deep NN with encoder-decoder architecture is leveraged for pixel-wise labeling of the predicted topology. Another encoding-decoding U-Net CNN architecture is presented in [24], providing up- and down-sampling operators based on training with large datasets. In [25] a multilevel topology optimisation is considered where the macroscale elastic structure is optimised subject to spatially varying microscale metamaterials. Instead of density, the parameters of the micromaterial are optimised in the iteration, using a single layer feedforward Gaussian basis function network as surrogate model for the elastic response of the microscale material.

A discussion on solving PDEs with the help of Neural Networks (NN) for instance of the Poisson equation and the steady NavierâĂŞStokes equations is provided in [7]. In a relatively new approach, a combination of Deep Learning and conventional topology optimisation, the Solid Isotropic Material with Penalisation (SIMP) was presented in [16], which could reduce the computational time compared to the classical approach. The authors use a similar method as [4] except that the underlying optimisation algorithm performs a mesh refinement after a fixed number of iterations. To improve this step, separately trained NNs are applied to the respective mesh in order to approximate the last steps of the optimisation on the corresponding mesh. The result then is projected to the next finer mesh and the procedure is repeated a fixed number of times. A SIMP density field topology optimisation is directly performed in [5]. The problem can be represented in terms of the NN activation function. Different beam problems comparable to our experiments are depicted. Fully connected DNN are used in [6] to represent implicit level set function describing the topology. For the optimisation, a system of parameterised ODEs is used. A two-stage NN architecture which by construction reduces the problem of structural disconnections is developed in [3]. Deep generative models for topology optimisation problems with varying design constraints and data scenarios are explored in [17]. In [1], direct predictions without any iteration scheme and also the nonlinear elastic setting are considered. Examples are only shown for a coarse mesh discretisation of the design domain. In [13], an NN assisted design method for topology optimisation is devised, which does not require any optimised data. A predictor NN provides the designs on the basis of boundary conditions and degree of filling as input data for which no optimisation training data is required.

The main goal of this paper is to devise new NN architectures that lower the computational burden of structural topology optimisation based on a continuous phase field description of the density in the design domain. In particular, the approach should be able to cope with adaptive mesh refinements during the optimisation process, which has shown to significantly improve the performance of the optimisation. Moreover, as a consequence of an efficient computation in a deterministic setting, a goal is to transfer the developed techniques to the stochastic setting for the risk-averse topology optimisation task. The general strategy is to combine conventional topology optimisation methods and NNs in order to reduce the number of required iteration steps within the optimisation procedure, increasing the overall performance.

The main achievements of this paper are two new NN architectures that are demonstrated to yield state-of-the-art numerical results with a much lower number of iterations than with a classical optimisation. Moreover, in contrast to several other works that are solely founded on the image level of topology, our architectures make use of a very versatile functional phase field description of the material distribution, which we have not seen in the literature with NNs. This also holds true for the stochastic risk-averse framework, which to our knowledge has not been considered with NN predictions yet. Another novelty is the mixture of a fine reference mesh and adaptive iteration meshes during optimisation.

Inspired by the work of [26] and [16], as a first new NN architecture we develop a new CNN approach and show that it can replicate the reference results of [9, 10]. In contrast to [16], we only have to train one NN for the entire optimisation despite mesh refinement being carried out in the iterative procedure. We subsequently extend this approach to the stochastic setting with risk-averse optimisation from [9]. Based on the CNN, we further extend the optimisation with a Long Short-Term Memory (LSTM) architecture as a second novel method. It encodes the change of the topology over several iteration steps, thus resulting in a more accurate prediction of the next gradient step.

In the numerical experiments it can be observed that the two presented architectures perform equally well as our reference implementation. However, fewer iteration steps are required (i.e., larger steps can be used) since the gradient step predictions seem to be better than when computed with a classical optimisation algorithm.

The structure of the paper is as follows. In Section 2, we introduce the underlying setting from [9, 10] and discuss the algorithms used for phase-field based topology optimisation. In this context, we introduce the linear elasticity model and derive a state equation, the adjoint equation and a gradient equation, whose joint solution constitutes the optimisation problem under consideration. In Section 3, we present two different architectures of the NNs approximating multiple steps of the gradient equation. We start with a CNN that is well suited for processing the discretised solutions of the equations from Section 2. This is then extended to a long short-term memory NN, which is able to process a sequence of these solutions at once and thus achieves a higher prediction quality. Since the data of the finite element simulation does not directly match the required structure of NNs, we provide a discussion of the data preparation for both architectures. Section 4 illustrates the practical performance of the developed NN architectures with a standard benchmark (a 2D bridge problem). The work ends with a summary and discussion of the results and some ideas for further research in Section 5. The appendix provides some background on the used problem, in particular the standard benchmark problem in Section A and the finite element discretisation in Section B. The implementation and codes for the generation of graphics and data to reproduce this work are publicly available[1].

2 Topology optimisation under uncertainties

We are concerned with the task of topology optimisation with respect to a state equation of linear elasticity. This problem becomes more involved when stochastic data is assumed. In our case, this concerns material properties and the forces acting on the designed structure. These translate into the engineering world as material imperfections or fluctuations and natural forces such as wind or ocean waves. Such random phenomena are modelled in terms of random fields that often are assumed to be Gaussian with certain mean and covariance.

It is instructive to first present the deterministic topology optimisation task, which we discuss in the following Section 2.1. Subsequently, in Section 2.2 we extend the model to exhibit random data, allowing an extension of the cost functional to also include the fluctuations of the data in terms of a risk measure. In our case, this is the so-called conditional value at risk (CVaR).

For the sake of a self-contained presentation, we provide all equations that lead to the actual optimisation problem, which is given in terms of a gradient that evolves a phase field. Thus, the entire problem formulation can be understood and the required extensions to obtain the risk-averse formulation become clear. However, in case that the reader is only interested in the proposed NN architectures, it

[1] `https://github.com/MarvinHaa/DeepNNforTopoOptisation`.

might be sufficient to just gloss over the most important parts of the problem definition, for which we provide a guideline as follows: The linear state equation in given in equation (2.1), leading to the weak form in equation (2.2) that is used for the computation of finite element solutions. These are required in the deterministic minimisation problem given in equation (2.4), which is solved iteratively by computing the gradient step defined by equation (2.6). A similar problem formulation, extended by an approximation of the CVaR risk measure, can be obtained in case of the risk-averse optimisation. This is given in equation (2.9) and can be solved iteratively with gradient steps defined by equation (2.11).

The presentation of this section is based on [9, 10] where the optimisation problem computes the distribution of material in a given design domain described by a continuous phase field depending on the realisation of the random parameters. The optimum of this problem maximises stiffness and at the same time minimises the volume of the material for the given data.

2.1 Deterministic model formulation

The goal is to determine an optimal distribution of a material (with density or probability) $m \in [0,1]$ in a compact design domain $D \subset \mathbb{R}^d,\ d \in \mathbb{N}$. We further assume that D satisfies sufficient regularity assumptions such that the PDE state equation exhibits a unique solution. The desired optimality of the task means that the resulting topology is as resilient (or stiff) as possible with respect to the deformation caused by the expected forces acting on it, which are described by a differentiable vector field $u : D \to \mathbb{R}^d$.

Definition 2.1. *The distribution of a material $m \in [0,1]$ in $D \subset \mathbb{R}^d$ is represented by a phase field $\varphi : D \to \mathbb{R}$ with $0 \leq \varphi(x) \leq 1$ for all $x \in D$, where $\varphi(x) = 1$ if there is material at position x and $\varphi(x) = 0$ if there is no material at position x. At the phase transitions we allow $0 < \varphi(x) < 1$ to ensure sufficient smoothness for phase shift. We call the evaluation of φ topology.*

Note that the actual topology is reconstructed in a post-processing step by choosing some threshold in $(0,1)$ to fix the interface between material and void phase of the phase field.

Linear elasticity model The state equation corresponding to the above problem is described by the standard linear elasticity model [22]. To define the material tensor, we first define the *strain displacement* (or strain tensor) $\mathcal{E} : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ by

$$\mathcal{E}(x) := \frac{1}{2}(\nabla u(x) + \nabla u^T(x)),$$

which specifies the displacement of the medium in the vicinity of position x. Moreover, a so-called blend function $\omega : \mathbb{R} \to \mathbb{R}$ is given by

$$\omega(x) := \max\{x^3, 0\},$$

ensuring a smooth transition between the phases. According to Hooke's law and by using the *Lamé coefficients* $\mu_{mat} > 0$ and $\lambda_{mat} > 0$, the *isotopic material tensor* $\sigma_{mat} : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ for the solid phase is given by

$$\sigma_{mat}(x) := 2\mu_{mat}\mathcal{E}(x) + \lambda_{mat}\mathrm{Tr}(\mathcal{E}(x))I.$$

This material tensor describes the acting forces between adjacent positions in the connected material, where λ_{mat} and μ_{mat} are two material parameters characterising the strain-stress relationship. For the void phase, to ensure solvability of the state equation in entire domain D, we define the tensor as a fraction of the material phases. More precisely, we set $\sigma_{\text{void}}(x) := \varepsilon^2\sigma_{\text{mat}}(x)$ with some small $\varepsilon > 0$. Hence, the *material tensor* (or stress tensor) $\sigma : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ is given by

$$\sigma(\varphi(x), u(x)) := \sigma_{\text{mat}}(u(x))\omega(\varphi(x)) + \sigma_{\text{void}}(u(x))\omega(1 - \varphi(x)).$$

Using the material tensor σ, a force with load $g \in \mathbb{R}^d$ (a pressure field) and the phase field φ, the displacement vector field u is described by the state equation of the standard linear elasticity model given by

$$\begin{aligned} -\operatorname{div}[\sigma(\varphi(x), u(x))] &= 0 && \text{for all } x \in D,\\ u(x) &= 0 && \text{for all } x \in \Gamma_D,\\ \sigma(\varphi(x), u(x)) &= g && \text{for all } x \in \Gamma_g,\\ u(x)\cdot n(x) &= 0 && \text{for all } x \in \Gamma_s,\\ \sigma(\varphi(x), u(x))\cdot n(x) &= 0 && \text{for all } x \in \Gamma_0 = \partial D \setminus (\Gamma_D \cup \Gamma_g \cup \Gamma_s). \end{aligned} \tag{2.1}$$

This implies that on boundary subspace $\Gamma_D \subset D$ the material is fixed while on $\Gamma_g \subset D$ the force g acts on the material. On the boundary $\Gamma_s \subset D$ the material is barred from movement in normal direction n. In the following, equality usually is to be understood in a pointwise way.

State equation The weak formulation of the state equation (2.1) can be formulated as: find $u \in H^1_{\Gamma_g}(D)$ such that

$$\int_D \sigma(\varphi, u)\mathcal{E}(v_u)\,\mathrm{d}\mu = \int_{\Gamma_g} g\cdot v_u\,\mathrm{d}\mu \qquad \forall\, v_u \in H^1_0(D), \tag{2.2}$$

where $H^1(D)$ is the usual Sobolev space and $\mathrm{d}\mu$ the Lebesgue measure and

$$H^1_{\Gamma_g}(D) := \{u \in H^1(D) \mid \sigma(\varphi, u) = g \text{ on } \Gamma_g\}$$

and

$$H^1_0(D) := \{v_u \in H^1(D) \mid v_u = 0 \text{ on } \Gamma_0\}.$$

These definitions are in particular used for the finite element discretisation described in Section B.

Adjoint equation To define the optimisation problem, we introduce the *Ginsburg-Landau functional* $E^\varepsilon : \mathbb{R} \to \mathbb{R}$, which serves as a penalty term for undesired variations and is defined by

$$E^\varepsilon(\varphi) = \int_D \frac{\varepsilon}{2}|\nabla\varphi|^2 + \frac{1}{2\varepsilon}\psi_0(\varphi)\,\mathrm{d}\mu,$$

where $|\cdot|$ is the Euclidean norm. This ensures that the solution to the optimisation problem can be interpreted as an actual smooth shape. The *double well functional* $\psi_0 : \mathbb{R}^d \to \mathbb{R}$ with $\psi_0(x) = (\varphi(x) - \varphi(x)^2)^2$ penalises values of φ that differ from 0 or 1 and the leading term limits the changes of φ. This results in the cost functional $J^\varepsilon : \mathbb{R}^d \to \mathbb{R}$ to be minimised,

$$J^\varepsilon(\varphi, u) = \int_{\Gamma_g} g\cdot u\,\mathrm{d}\mu + \gamma E^\varepsilon(\varphi), \quad \gamma > 0. \tag{2.3}$$

The adaptivity parameter γ controls the weight of the interface penalty and hence has a direct influence on the minimum respectively the characteristics of the resulting shape of φ. In fact, γ is chosen adaptively to avoid non-physical or highly porous topologies, see [9]. Additionally, we require the volume constraint $\int_D \varphi \,\mathrm{d}\mu = m|D|$ with $m \in [0,1]$ to limit the amount of overall material.

The (displacement) state u from equation (2.3) is obtained by solving the state equation (2.2), which is used in the optimisation problem

$$\text{minimize } J^\varepsilon(\varphi, u) \text{ over } \varphi \in H^1(D) \tag{2.4}$$
$$\text{s.t. equation (2.2) holds, } 0 \le \varphi(x) \le 1 \text{ for all } x \in D \text{ and } \int_D \varphi(x)\,\mathrm{d}\mu = m|D|.$$

The *Allen-Cahn gradient flow approach* is used to determine the solution φ for which the adjoint problem of equation (2.4) is used to avoid the otherwise more costly calculation. It is shown in [9] that for J^ε the corresponding adjoint problem can be formulated as: find $p \in H^1(D)$ such that

$$\int_D \sigma(\varphi, p))\mathcal{E}(v_p)\,\mathrm{d}\mu = \int_{\Gamma_g} g \cdot v_p \,\mathrm{d}\mu \qquad \forall\, v_p \in H_0^1(D), \tag{2.5}$$

which is identical to the state equation. Hence, the respective adjoint solution p is equal to the solution u of equation (2.2) and no additional system has to be solved.

Gradient equation With the solutions u respectively p one gradient step with adaptive step size τ can be characterised by the unique solution $(\varphi, \lambda) \in H^1(D) \times \mathbb{R}$ such that, for all $(v_\varphi, v_\lambda) \in H_0^1(D) \times \mathbb{R}$,

$$\begin{aligned}\frac{\varepsilon}{\tau}\int_D (\varphi^* - \varphi_n)v_\varphi \,\mathrm{d}\mu + \varepsilon\gamma \int_D \nabla\varphi^* \cdot \nabla v_\varphi \,\mathrm{d}\mu + \frac{\gamma}{\varepsilon}\int_D \frac{\partial}{\partial\varphi}\psi_0(\varphi_n)v_\varphi \,\mathrm{d}\mu \\ - \int_D \frac{\partial}{\partial\varphi}\sigma(p, \varphi_n)v_\varphi \mathcal{E}(u)\,\mathrm{d}\mu + \int_D \lambda v_\varphi \,\mathrm{d}\mu + \int_D (\varphi^* - m)v_\lambda \,\mathrm{d}\mu = 0.\end{aligned} \tag{2.6}$$

The restriction on $0 \le \varphi \le 1$ for all $x \in D$ is realised by $\varphi(x) := \min\{\max\{0, \varphi^*(x)\}, 1\}$ in every iteration step. For the calculation of the minimum of equation (2.4), the state equation (2.1), the adjoint equation (2.5) and subsequently the gradient equation (2.6) are solved iteratively until φ converges. We always assume that solutions u and φ exist, which in fact can be observed numerically. The proposed procedure is described by Algorithm 4 where the solution of the integral equations takes place on a discretisation of D. The algorithm solves the state, adjoint and gradient equations in a loop until the solutions of the gradient equations only change slightly. The discretisation mesh is subsequently refined and the iterative process is restarted on this adjusted discretisation.

2.2 Stochastic model formulation

In the stochastic setting the Lamé coefficients (determining the material properties) $\lambda_{\text{mat}} : \Omega \to \mathbb{R}^+$ and $\mu_{\text{mat}} : \Omega \to \mathbb{R}^+$ and the load scenarios $g : \Omega \to \mathbb{R}^d$ are treated as random variables on some probability space (Ω, P). The randomness of the data is inherited by the solution of the state equation as well as the adjoint equation. As a result, the gradient step can be considered as a random distribution, see again [9, 10]. The goal is to minimise the functional for the expected value of φ as well as for particularly unlikely events. For the formulation of an adequate risk-averse cost functional, we introduce the *conditional value at risk* (CVaR). The CVaR, a common quantity in financial mathematics, is defined for a random variable X by

$$\mathrm{CVaR}_\beta[X] := \mathbb{E}[X \ \mathbb{1}_{\{X > \mathrm{VaR}_\beta[X]\}}],$$

with $\mathrm{VaR}_\beta[X] := \inf\{t \in \mathbb{R} | P(X \leq t) \geq \beta\}$ and $1 > \beta \geq 0$. It characterises the expectation of the β-tail quantile distribution of X, hence accounting for bad outliers that may occur with low probability. The *stochastic state equation* can be formulated analogously to equation (2.5) in the deterministic setting.

Adjoint equation For the risk-aware version of equation (2.3) with respect to the CVaR parameter β, we define the cost $J_\beta^\varepsilon(\varphi) : \mathbb{R}^d \to \mathbb{R}$ by

$$J_\beta^\varepsilon(\varphi) = \mathrm{CVaR}_\beta\Big[\int_{\Gamma_g} g \cdot u \,\mathrm{d}\mu\Big] + \gamma E^\varepsilon(\varphi), \quad \gamma > 0. \tag{2.7}$$

In the special case $\beta = 0$, the CVaR is nothing else than the mean, i.e.,

$$J_0^\varepsilon(\varphi) = \mathbb{E}\Big[\int_{\Gamma_g} g \cdot u \,\mathrm{d}\mu\Big] + \gamma E^\varepsilon(\varphi).$$

This results in the *stochastic minimisation problem* analogous to equation (2.4) given by

$$\text{minimise } J_\beta^\varepsilon(\varphi) \text{ over } \varphi \in H^1(D) \tag{2.8}$$

$$\text{s.t. equation (2.2) holds a.s., } 0 \leq \varphi(x) \leq 1 \text{ for all } x \in D \text{ and } \int_D \varphi \,\mathrm{d}\mu = m|D|.$$

Following [9], the CVaR can be approximated in terms of the plus function. The solution of equation (2.8) can hence be rewritten as

$$\min_{\varphi \in H^1(D)} J_\beta^\varepsilon(\varphi) = \min_{\varphi \in H^1(D), t \geq 0} \Big(t + \frac{1}{1-\beta}\mathbb{E}\Big[\Big(\int_{\Gamma_g} g \cdot u \,\mathrm{d}\mu\Big)_+\Big] + \gamma E^\varepsilon(\varphi)\Big). \tag{2.9}$$

An obvious approach is to solve this optimisation problem by Monte Carlo simulations, i.e., for each iteration step $n \in \mathbb{N}$ with evaluation of u_n, state equations (2.1) have to be solved for different parameter realisations. The associated adjoint problem to equation (2.9) reads

$$\int_D \sigma(\varphi, p)\mathcal{E}(v_p) \,\mathrm{d}\mu = \begin{cases} 0, & \text{if } \int_{\Gamma_g} gu \,\mathrm{d}\mu - t \leq 0 \\ \int_{\Gamma_g} (1-\beta)^{-1} gu \,\mathrm{d}\mu, & \text{else} \end{cases} \quad \text{a.s.} \tag{2.10}$$

Consequently, the solution of equation (2.10) is given by

$$p = \begin{cases} 0, & \text{if } \int_{\Gamma_g} gu \,\mathrm{d}\mu - t \leq 0 \\ (1-\beta)^{-1} u, & \text{else} \end{cases} \quad \text{a.s.}$$

Gradient equation Analogous to the deterministic approach, the gradient can be defined corresponding to equation (2.9) by the solution $(\varphi, \lambda, t) \in H^1(D) \times \mathbb{R} \times \mathbb{R}$, such that for all $(v_\varphi, v_\lambda, v_t) \in H_0^1(D) \times \mathbb{R} \times \mathbb{R}$ the following equation holds,

$$\begin{aligned} 0 = &\frac{\varepsilon}{\tau_\varphi}\int_D (\varphi^* - \varphi_n) v_\varphi \,\mathrm{d}\mu + \frac{\varepsilon}{\tau_t}\int_D (t - t_n) v_t \,\mathrm{d}\mu + \int_D \lambda v_\varphi \,\mathrm{d}\mu + \int_D (\varphi^* - m) v_\lambda \,\mathrm{d}\mu \\ &+ \varepsilon\gamma \int_D \nabla\varphi^* \cdot \nabla v_\varphi \,\mathrm{d}\mu + \frac{\gamma}{\varepsilon}\int_D \frac{\partial}{\partial\varphi}\psi_0(\varphi_n) v_\varphi \,\mathrm{d}\mu - \int_D \frac{\partial}{\partial\varphi}\sigma(p, \varphi_n) v_\varphi \mathcal{E}(u) \,\mathrm{d}\mu \\ &+ \begin{cases} \int_D v_t \,\mathrm{d}\mu, & \text{if } \int_{\Gamma_g} gu \,\mathrm{d}\mu - t \leq 0 \\ \int_D (1 - \frac{1}{1-\beta}) v_t \,\mathrm{d}\mu, & \text{else} \end{cases} \quad a.s. \end{aligned} \tag{2.11}$$

The actual solution for one gradient step of the minimisation problem in equation (2.8) respectively (2.9) follows from

$$\varphi \approx \frac{1}{S}\sum_{i=1}^{S}\varphi_i^*, \tag{2.12}$$

where $S \in \mathbb{N}$ is the number of samples of the Monte Carlo simulation. The larger β is chosen, the larger t becomes and thus the number of evaluations of u for which $\int_{\Gamma_g} gu \,\mathrm{d}\mu - t \leq 0$ holds true increases. In order to ensure a valid simulation of equation (2.12), N must be chosen sufficiently large so that an adequate number of evaluations of u in each gradient step fulfils condition $\int_{\Gamma_g} gu \,\mathrm{d}\mu - t > 0$. The described procedure is depicted in Algorithm 5 where the optimisation process is basically the same as in Algorithm 4. The central difference is that $N \in \mathbb{N}$ realisations of the optimisation problem have to be computed in each iteration. In practice, these are solved in parallel for N different $\omega \in \Omega$ and the results are then averaged. The large computational efforts caused by the slow Monte Carlo convergence are alleviated by the neural network based machine learning approaches presented in Section 3. In particular, gradient steps for arbitrary parameter realisations can be evaluated very efficiently and significantly fewer iterations (i.e., optimisation iterations) are required.

3 Neural Network architectures

In modern scientific and engineering computing, machine learning techniques have become indispensable in recent years. The central goal of this work is to devise neural network architectures to facilitate an efficient computation of the risk-averse stochastic topology optimisation task. In this section, we develop two such architectures. The first one described in Section 3.1 is based on the popular convolutional neural networks (CNN) that have originally been designed for the treatment of image data. In Section 3.2, a classical long short-term memory architecture (LSTM) is adapted to predict the gradient step.

The usage of deep neural networks with topology optimisation tasks as have already been examined in [16] and [4]. However, in contrast to other approaches, our architecture aims at a single NN that can be trained to handle arbitrarily fine meshes in terms of what is required during the topology optimisation process. More precisely, we want to find a NN that predicts the gradient step $\varphi_{n+k} \in \mathbb{R}^{|V(\mathcal{T}_m)|}$ from equation (2.6) discretised on an arbitrarily fine mesh T_m at an arbitrary iteration step $n \in \mathbb{N}$ with given $k \in \mathbb{N}$, for $\mathcal{N}^k : \mathbb{R}^{1+d\times|V(\mathcal{T}_m)|} \to \mathbb{R}^{|V(\mathcal{T}_m)|}$ such that

$$\mathcal{N}^k([\varphi_n, u_{n+1}]) = \varphi_{n+k}. \tag{3.1}$$

Thus the total number of iterations required for the topology optimisation iteration should ideally be reduced, resulting in improved practical performance. For the sake of a convenient presentation, we consider all other coefficients of equation (2.6) as constant in the following analyses. Alternatively, one would have to increase the complexity in the number of degrees of freedom which are the weights describing the NN as well as the required training data. It can be assumed that with more information in the form of coefficients provided to the NN during training the accuracy of the resulting approximation of φ_n increases. Within the optimisation procedure, the actual calculation of the gradient step given in equation (2.6) is done on the basis of variable coefficients (e.g τ and γ).

Since the discretisation $\varphi_n \in \mathbb{R}^{|V(\mathcal{T}_m)|}$ can be rewritten rather easily in tensor form, which represents the input of a CNN, this is the first architecture we consider in the next section.

3.1 Topology Convolutional Neural Networks (TCNN)

When using a visual representation of topologies as images (as they can be generated as output of a finite element simulation), the solution of equation (2.6) can be transferred easily to the data structures that are used in CNNs. Consequently, predicting the gradient step with a CNN can be understood as a projection of the optimisation problem into a pixel-structured image classification problem. Here we assume that the calculation of the learned gradient step is encoded in the weights that characterise the NN.
In principle, the structure of a classical CNN consists of one or more convolutional layers followed by a pooling layer. This basic processing unit can repeat itself as often as desired. If there are at least three repetitions we speak of a deep CNN and a deep learning architecture. In the convolutional layers a convolution matrix is applied to the input. The pooling layers are to be understood as a dimensional reduction of their input. Although common for image classification tasks, pooling layers are not used in the presented architecture.

3.1.1 TCNN architecture

We follow the presentation of the `pytorch` documentation [19]. The input of a layer of the CNN architecture is a tensor $\mathcal{I} \in \mathbb{R}^{S\times C_{\text{in}}\times H\times W}$. Here, $S \in \mathbb{N}$ is the number of input samples, in our case the evaluations of φ and u as presented in Section 3.1.2. It is therefore possible to calculate the gradient step φ_n from equation (2.6) for several different loads g simultaneously. This way, Monte Carlo estimates become very efficient. $C_{in} \in \mathbb{N}$ corresponds to the number of input channels and each channel represents one dimension of an input (φ or u). $H \in \mathbb{N}$ and $W \in \mathbb{N}$ provide information about the dimension of the discretisation of the space D. The output of one CNN layer is specified by $\mathcal{O} \in \mathbb{R}^{S\times C_{out}\times H\times W}$. For fixed $s \leq S$ and $i \leq C_{\text{out}} \in \mathbb{N}$ with C_{out} the number of output channels is given as

$$\mathcal{O}_{s,i}(\mathcal{I}_s) = b_i + \sum_{k=1}^{C_{in}} \mathcal{W}_{i,k} * \mathcal{I}_{s,k}. \tag{3.2}$$

Here, $*$ denotes the cross-correlation operator, $b \in \mathbb{R}^{C_{out}\times H\times W}$ and $\mathcal{I}_{s,k}$ with $s \leq S, k \leq C_{\text{in}}$ is a cutout of $\mathcal{I}$. The weight tensor $\mathcal{W} \in \mathbb{R}^{C_{\text{out}}\times C_{in}\times H_K\times W_K}$ determines the dimensions of the kernel (or convolution matrix) of the layers with $H_K, W_K \in \mathbb{N}$.

For simplicity we henceforth assume $S = 1$ unless otherwise specified. In particular, the entries of the weight tensor $\mathcal{W}$ are parameters that are optimised during the training of the CNN. Depending on the architecture of the CNN, an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ evaluated elementwise can additionally be applied to equation (3.2).

Definition 3.1. *Let $b \in \mathbb{R}^{C_{\mathit{out}}\times H\times W}$ and $\mathcal{W} \in \mathbb{R}^{C_{\mathit{in}}\times C_{\mathit{out}}\times H_K\times W_K}$ with $L, H, W, H_K, W_K, C_{\mathit{in}}, C_{\mathit{out}} \in \mathbb{N}$ be given by one parameter vector $\theta \in \mathbb{R}^d$ with*

$$d = C_{\mathit{out}} \cdot H \cdot W + C_{\mathit{out}} \cdot C_{\mathit{in}} \cdot H_K \cdot W_K.$$

Furthermore, let σ be a continuously differentiable activation function. We call a function

$$\mathit{Conv}(\,\cdot\,;\theta) : \mathbb{R}^{C_{\mathit{in}}\times H\times W} \to \mathbb{R}^{C_{\mathit{out}}\times H\times W} \tag{3.3}$$

a convolution layer with activation function σ if it satisfies

$$\mathit{Conv}(\mathcal{I};\theta)_i = \sigma\Big(b_i + \sum_{k=1}^{C_{in}} \mathcal{W}_{i,k} * \mathcal{I}_k\Big) \tag{3.4}$$

with $i = 1, \dots, C_{out}$.

A sequential coupling of this layer structure provides the framework for the CNN. Specifically, for $i^L = 1, \dots, C^L_{out} \in \mathbb{N}$,

$$\begin{aligned}\mathrm{Conv}(\,\cdot\,;\theta^{L-1})^L_{i^L} \circ \mathrm{Conv}(\,\cdot\,;\theta^{L-1})^{L-1} \circ \dots \circ \mathrm{Conv}(\mathcal{I};\theta^1)^1 = \\ \sigma^L\Big(b^L_{c_{\mathrm{out}}} + \sum_{k_L=1}^{C^{L-1}_{\mathrm{out}}} \mathcal{W}^L_{c_{\mathrm{out}},k_L} * \sigma^{L-1}\Big(b^{L-1}_{c_{\mathrm{out}}} + \sum_{k_{L-1}=1}^{C^{L-2}_{\mathrm{out}}} \mathcal{W}^{L-1}_{c_{\mathrm{out}},k_{L-1}} * \dots \\ \dots * \sigma^1\Big(b^1_{c_{\mathrm{out}}} + \sum_{k_1=1}^{C_{\mathrm{in}}} \mathcal{W}^1_{c_{\mathrm{out}},k_1} * \mathcal{I}_{k_1}\Big) \dots\Big)\Big)_{i^L}. \qquad (3.5)\end{aligned}$$

Definition 3.2 (CNN architecture). *Let* $\mathcal{W}^1 \in \mathbb{R}^{C_{in} \times C^1_{out} \times H_K \times W_K}$, $\mathcal{W}^l \in \mathbb{R}^{C^{l-1}_{out} \times C^l_{out} \times H_K \times W_K}$ *for* $1 < l \leq L$ *and* $b^l \in \mathbb{R}^{C^l_{out} \times H \times W}$ *for* $l \leq L$ *with* $L, H, W, H_K, W_K, C_{in}, C^1_{out}, \dots, C^L_{out} \in \mathbb{N}$ *be given by some parameter vector* $\theta \in \mathbb{R}^d$ *with*

$$d = \underbrace{C^1_{out} * (C_{in}(H_K \cdot W_K) + (H \cdot W))}_{\text{Dimension of } \mathcal{W}^1 \text{ and } b^1} + \underbrace{\sum_{l=2}^{L} C^l_{out} * (C^{l-1}_{out}(H_K \cdot W_K) + (H \cdot W))}_{\text{Dimension of } \mathcal{W}^l \text{ and } b^l \text{ for } 2 \leq l \leq L}.$$

Furthermore, let $\sigma^1, \dots, \sigma^L$ *be given continuously differentiable activation functions. We call a NN of the form of equation* (3.5) *an* L*-layer topology convolutional neural network (TCNN) and characterise it as the mapping*

$$\mathcal{N}_{CNN}(\,\cdot\,;\theta) : \mathbb{R}^{C_{in} \times H \times W} \to \mathbb{R}^{C_{out} \times H \times W}.$$

The approximation of $\mathcal{N}_{\mathrm{CNN}}$ is hence determined by its parameter vector θ. For general CNNs, the dimension $H \times W$ does not have to be constant across the different layers. The same holds true for the dimensions $H_K \times W_K$ of the kernel matrices. In fact, before implementing the convolution, we embed each channel of our input in a $(H + \lfloor \frac{H_K}{2} \rfloor) \times (W + \lfloor \frac{W_K}{2} \rfloor)$ space to preserve the dimension in the output.

Example 3.1. *The following specific TCNN has proved to be the most suitable for integration into Algorithm 4 for the selections of hyperparameters we have investigated. The architecture is given as a* $L = 6$ *layer TCNN with* $C_{in} = 3$ *input channels,* $C^l_{out} = 15$ *for* $1 < l < 5$ *hidden channel,* $C^6_{out} = 1$ *output channel and kernel size* $H_K = W_K = 3$ *as well as trained weights described by* $\theta \in \mathbb{R}^{8806}$ *which determine the mapping by*

$$\mathcal{N}_{CNN}(\,\cdot\,;\theta) : \mathbb{R}^{3 \times 201 \times 101} \to \mathbb{R}^{1 \times 201 \times 101}, \qquad (3.6)$$

with activation function $\sigma^6(x) := \min\{\max\{x, 0\}, 1\}$. *In contrast to many standard architectures, only the activation function of the output layer is not the identity. We chose* $\mathbb{R}^{3 \times 201 \times 101}$ *as input space in anticipation of the setting in Example 3.2, reflecting our mesh choice to discretise domain* $D = [-1, 1] \times [0, 1]$ *with* 201×101 *nodes, which for first order finite elements then is the dimension of the discrete functions* u *and* φ.

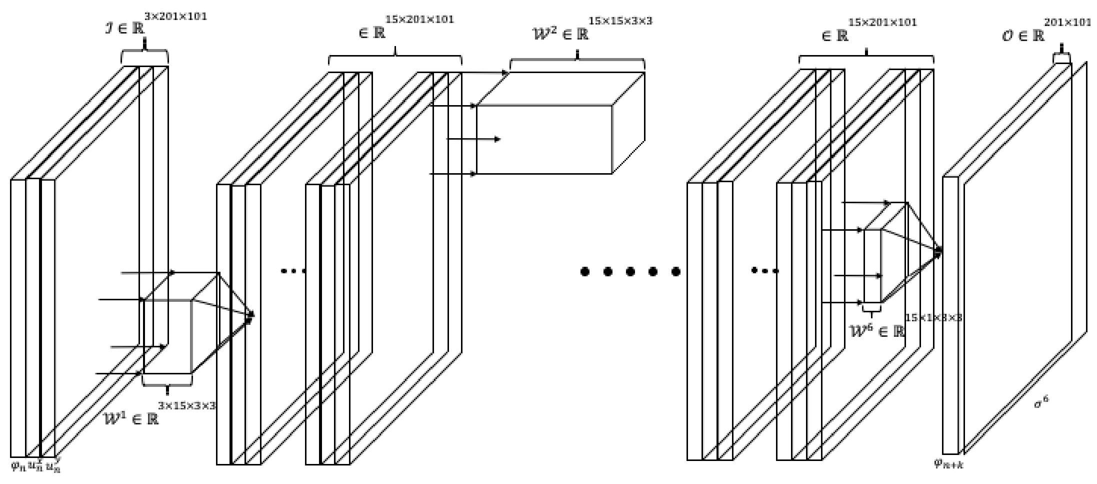

$\mathcal{I} \in \mathbb{R}^{3\times201\times101}$
$\in \mathbb{R}^{15\times201\times101}$
$\mathcal{W}^2 \in \mathbb{R}^{15\times15\times3\times3}$
$\in \mathbb{R}^{15\times201\times101}$
$\mathcal{O} \in \mathbb{R}^{201\times101}$
$\mathcal{W}^1 \in \mathbb{R}^{3\times15\times3\times3}$
$\mathcal{W}^6 \in \mathbb{R}^{15\times1\times3\times3}$
σ^6
φ_n u_n^x u_n^y
φ_{n+k}

Figure 1: Visualisation of the TCNN from Example 3.1.

3.1.2 Data preparation

On the algorithmic level, our goal is to replace the computationally costly lines 5 and 6 of all $c_m \in \mathbb{N}$ loop iterations of Algorithm 4 with a TCNN. This is not directly possible (at least for a TCNN) since the input space $\mathbb{R}^{C_{\text{in}}\times H\times W}$ of a TCNN does not match the mesh $\mathcal{T}_m$ on which the finite element discretisation and thus the optimisation of φ_n takes place in the current optimisation step t. It is hence necessary to project the evaluation of φ onto the format of a CNN. For this we define a transformation between $\mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{in}}}$ and the input tensor $\mathbb{R}^{H\times W\times C_{in}}$ of $\mathcal{N}_{\text{CNN}}$. As described in Section B, we do not assume that the mesh $\mathcal{T}_m$ stays fixed in the optimisation algorithm and we instead generate a sequence of different meshes $\mathcal{T}_m$ by some adaptive mesh refinement, which has led to significant efficiency improvements in [10]. To get unique transformations between the discretisation finite element space and the input space of the NN, one can interpolate the current solutions of φ_n and u_{n+1} from $\mathcal{T}_m$ onto a constant reference mesh $\mathcal{T}_{\text{const}}$ by polynomial interpolations

$$\begin{aligned} p &: \mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{in}}} \to \mathbb{R}^{H\cdot W\times C_{\text{in}}}, \\ q &: \mathbb{R}^{H\cdot W\times C_{\text{out}}} \to \mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{out}}}. \end{aligned}$$

Hence, during the optimisation, the current solutions are interpolated via the operator p to the reference mesh, rendering the prediction independent from the actual adaptive mesh. After the $\mathcal{N}_{\text{CNN}}$ prediction of the gradient step on the reference mesh, it is mapped back to the actual computation mesh via q.

Consequently, we define the reference mesh $\mathcal{T}_{\text{const}} = (V(\mathcal{T}_{\text{const}}), E(\mathcal{T}_{\text{const}}))$ with vertices V and edges E as a graph such that $|V(\mathcal{T}_{\text{const}})| = H\cdot W$. Each node $v_i \in V(\mathcal{T}_{\text{const}})$ corresponds to the values of $\varphi_n^i = \varphi_n(v_i) \in \mathbb{R}$ and $u_n^i = u_n(v_i) \in \mathbb{R}^d,\ i \le H\cdot W = |V(\mathcal{T}_{\text{const}})|$ at node v_i. The features of the nodes can hence be interpreted as rows of a feature matrix,

$$\widetilde{\mathcal{I}_n} = \begin{bmatrix} \varphi_1 & u_1 \\ \vdots & \vdots \\ \varphi_{n_{H\cdot W}} & u_{n_{H\cdot W}} \end{bmatrix} \in \mathbb{R}^{H\cdot W\times C_{in}}. \tag{3.7}$$

The structure of $\mathcal{T}_{\text{const}}$ is illustrated in Figure 2. One can now define a transformation between $\mathbb{R}^{H\cdot W\times C_{in}}$

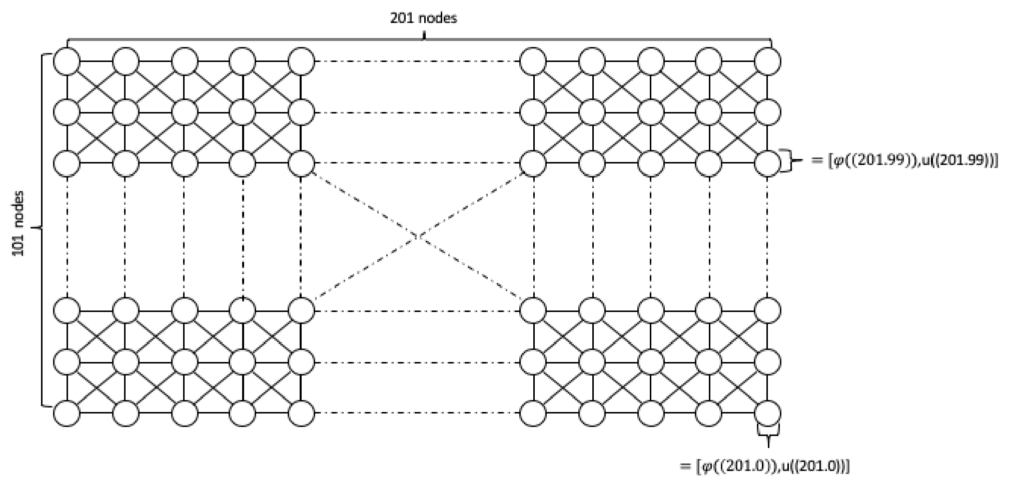


Figure 2: $\mathcal{T}_{\text{const}}$ for $\Phi_{C_{\text{in}}} : \mathbb{R}^{3\times 101\cdot 201} \to \mathbb{R}^{1\times 201\times 101}$ to transform $\mathcal{N}_{\text{TCNN}}$ from Example 3.1.

and $\mathbb{R}^{C_{in}\times H\times W}$ by

$$\begin{aligned}\Phi_{C_{in}} &: \mathbb{R}^{H\cdot W\times C_{\text{in}}} \to \mathbb{R}^{C_{\text{in}}\times H\times W},\\ \Phi_{C_{\text{out}}} &: \mathbb{R}^{C_{\text{out}}\times H\times W} \to \mathbb{R}^{H\cdot W\times C_{\text{out}}}.\end{aligned} \tag{3.8}$$

Hence, the approximation of the gradient step 6 of Algorithm 4 is basically a coupling of the mappings p, Φ and $\mathcal{N}_{CNN}$, namely

$$\begin{aligned}&\mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{in}}} \xrightarrow{p} \mathbb{R}^{H\cdot W\times C_{\text{in}}} \xrightarrow{\Phi_{C_{\text{in}}}} \mathbb{R}^{C_{\text{in}}\times H\times W} \xrightarrow{\mathcal{N}_{\text{CNN}}}\\ &\qquad \xrightarrow{\mathcal{N}_{\text{CNN}}} \mathbb{R}^{C_{\text{out}}\times H\times W} \xrightarrow{\Phi_{C_{\text{out}}}} \mathbb{R}^{H\cdot W\times C_{\text{out}}} \xrightarrow{q} \mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{out}}}.\end{aligned} \tag{3.9}$$

Example 3.2 (Illustrating the TCNN). *The NN given by the coupling of functions in equation* (3.9) *with $\mathcal{N}_{CNN}$ given as in Example 3.1 can be described by*

$$\mathcal{N}^k_{\textit{CNN}}(\,\cdot\,;\theta) : \mathbb{R}^{|V(\mathcal{T}_m)|\times 3} \to \mathbb{R}^{|V(\mathcal{T}_m)|}, \tag{3.10}$$

with

$$\begin{bmatrix}\varphi_n & u^x_{n+1} & u^y_{n+1}\end{bmatrix} \mapsto \begin{bmatrix}\varphi_{n+k}\end{bmatrix} \tag{3.11}$$

for $\varphi_n \in \mathbb{R}^{|V(\mathcal{T}_m)|}$ and $u_n = (u^x_n, u^y_n),\ u^x_n, u^y_n \in \mathbb{R}^{|V(\mathcal{T}_m)|}$ defined on $\mathcal{T}_m$. Hence, this NN can be applied directly to the finite element discretisations φ_n and u_n used in Algorithms 4 and 5.

With the TCNN from the example in equation (3.11) we have extended Algorithm 4. More precisely, we have inserted a NN approximation $\mathcal{N}^k_{\text{CNN}}(\varphi_n, u_{n+1};\theta)$ in each of the $c_m \in \mathbb{N}$ steps, which predicts k iteration steps by just one evaluation. For this, the sequence c_m hast to be defined in advance. We leave it to future research to adaptively control the sequence c_m dynamically within the optimisation algorithm. This extension of Algorithm 4 is described by Algorithm 1. In an analogous way, we also extend Algorithm 5 by the TCNN given in equation (3.11). In particular, we are able to evaluate all samples $S \in \mathbb{N}$ in parallel by adding additional sample dimensions to the input tensor of the TCNN given in equation (3.2). This procedure is illustrated in Algorithm 2. Again, the parameter c_m has to be chosen in advanced.

Algorithm 1: Deterministic optimisation algorithm with TCNN approximated gradient step.

```
Input: mesh $\mathcal{T}_0,\ \mathcal{T}_{const}$ and initial values $\varphi_0$ sequence $c_m \in \mathbb{N}$ and $j = 0 \in \mathbb{N}$
for $m = 0, 1, \dots$ until converged do
    for $n = 0, 1, \dots$ until converged do
        solve state equation on mesh $\mathcal{T}_m \Rightarrow u_{n+1}$
        if $j = c_m$ then
            interpolate $\varphi_n$ onto $\mathcal{T}_{\text{const}}$ project $u_{n+1}, \varphi_n$ to $\mathbb{R}^{d+1\times H\times W}$
            evaluate $\mathcal{N}^k_{\text{CNN}}(\varphi_n, u_{n+1}; \theta)$ like in equation (3.9) $\Rightarrow \varphi_{n+k}$
            project $\varphi_{n+k}$ to $\mathcal{T}_{\text{const}}$
            interpolate $\varphi_{n+k}$ onto $\mathcal{T}_m$
            $j = 1$
        else
            solve adjoint equation on mesh $\mathcal{T}_m \Rightarrow p_{n+1}$
            solve gradient equation on mesh $\mathcal{T}_m \Rightarrow \varphi^*_{n+1}$
            project $\varphi^*_{n+1}$ to $[0, 1] \Rightarrow \varphi_{n+1}$
            $j = j + 1$
        end
    end
    adapt mesh according to Section B $\Rightarrow \mathcal{T}_{m+1}$
end
```

3.2 Topology long short-term memory Neural Networks (TLSTM)

One possible approach to improve the prediction of φ_n using an NN is to provide the classifier not just one tuple (φ_n, u_{n+1}) as an input but to have it process a larger amount of information by a sequence of these tuples of the last $T \in \mathbb{N}$ iteration steps, i.e.,

$$\Big((\varphi_{n-T}, u_{n-T+1}), (\varphi_{n-T+1}, u_{n-T+2}), \dots, (\varphi_n, u_{n+1})\Big).$$

By this the shift of the phase field or the change of the topology φ_n over time is also transferred as input to the NN. The sequence prediction problem considered in this case differs from the single step time prediction in the sense that the prediction target is now a sequence that contains both spatial and temporal information. Theoretically, this information can also be learned directly from the NN. However, in practice it is more effective to adapt the architecture to the information we have in advance (in our case with respect to the time dependency) to achieve better results. An NN that allows exactly this is a recurrent Neural Networks (RNN). Unfortunately, standard RNNs often suffer from the vanishing gradient problem [14, 15] which we try to prevent right from the start. Therefore, we build on the special RNN concept of a Long Short-Term Memory (LSTM) in the context of our problem, which is more robust against the vanishing gradient issue and provides promising results, especially in the analysis of time series. For a background on time series analysis and the review of different methods, we refer to the survey article [11]. In practice, time series are usually stored as one-dimensional sequences in vector format. Consequently, there is no out-of-the-box LSTM layer implementation for structures like the input tensor we require in equation (3.2). Nevertheless, we still do not want to abandon the mechanism of convolution within the NN in order to keep the structural information of φ and u. A LSTM layer with convolutional structure can be constructed by replacing the matrix vector multiplication within a standard LSTM layer by convolutional layers. The unique selling point of an LSTM according to [15] is its cell-gate architecture, which mitigates the vanishing gradient problem. More precisely, it consists of

Algorithm 2: Stochastic optimisation algorithm with TCNN approximated gradient step.

```
Input: mesh $\mathcal{T}_0,\ \mathcal{T}_{\text{const}}$ and initial values $\varphi_0$, sequence $c_m \in \mathbb{N}$ and $j = 0 \in \mathbb{N}$
for $m = 0, 1, \dots$ until converged do
    for $n = 0, 1, \dots$ until converged do
        for $i = 1, \dots, N$ do
            sample $g(\omega_i), \lambda_{mat}(\omega_i), \mu_{mat}(\omega_i)$
            solve state equation on mesh $\mathcal{T}_m \Rightarrow u_{n+1}(\omega_i)$
            solve adjoint equation on mesh $\mathcal{T}_m \Rightarrow p_{n+1}(\omega_i)$
            if $j = c_m$ then
                interpolate $\varphi_n(\omega_i)$ onto $\mathcal{T}_{\text{const}}$
                project $u_{n+1}(\omega_i), \varphi_n(\omega_i)$ to $\mathbb{R}^{d+1\times H\times W}$
                evaluate $\mathcal{N}^k_{\text{CNN}}(\varphi_n(\omega_i), u_{n+1}(\omega_i); \theta)$ like in equation (3.9) $\Rightarrow \varphi_{n+k}(\omega_i)$
                project $\varphi_{n+k}(\omega_i)$ to $\mathcal{T}_{\text{const}}$
                interpolate $\varphi_{n+k}(\omega_i)$ onto $\mathcal{T}_m$
                $j = 1$
            else
                solve gradient equation on mesh $\mathcal{T}_m \Rightarrow \varphi^*_{n+1}(\omega_i)$
            end
        end
        compute the mean $\hat{\varphi}_{n+1} = \frac{1}{N}\sum_{i=1}^{N} \varphi^*_{n+1}(\omega_i)$
        project $\hat{\varphi}_{n+1}$ to $[0, 1] \Rightarrow \varphi_{n+1}$
        adapt mesh according to Section B $\Rightarrow \mathcal{T}_{m+1}$
        $j = j + 1$
    end
    adapt mesh according to Section B $\Rightarrow \mathcal{T}_{m+1}$
end
```

a “memory cell” $c_t : \mathbb{R}^{4\times d^t_{\text{in}}} \to \mathbb{R}^{d^t_{\text{out}}}$ that serves as an accumulator of the current state $t \leq T,\ t \in \mathbb{N}$, in the processed sequence. The information capacity of the last status c_{t-1} within c_t is controlled by the activation of the so-called “forget gate” $f_t : \mathbb{R}^{3\times d^t_{\text{in}}} \to \mathbb{R}^{d^t_{\text{out}}}$. The information capacity of the input state $x_t \in \mathbb{R}^{d^t_{\text{in}}}$ is controlled by the activation of the input gate i_t. Which information (or whether any at all) gets transferred from memory cell c_t to state $h_t : \mathbb{R}^{2\times d^t_{\text{in}}} \to \mathbb{R}^{d^t_{\text{out}}}$ is in turn controlled by the activation of the output gate o_t. From a technical point of view, the gates can be understood as learning forward layers.

3.3 TLSTM architecture

An ordinary LSTM layer to generate complex sequences with long-range structure as presented in [12] corresponds to the described logic above and can be formulated numerically for a sequence of one-dimensional input state $x_t \in \mathbb{R}^{d^t_{\text{in}}}$ and output vector $h_t \in \mathbb{R}^{d^t_{\text{out}}}$ as an equation system

$$\begin{aligned}
i_t(x_t, h_{t-1}, c_{t-1}) &= \sigma(\mathcal{W}_{xi}x_t + \mathcal{W}_{hi}h_{t-1} + \mathcal{W}_{ci} \odot c_{t-1} + b_i),\\
f_t(x_t, h_{t-1}, c_{t-1}) &= \sigma(\mathcal{W}_{xf}x_t + \mathcal{W}_{hf}h_{t-1} + \mathcal{W}_{cf} \odot c_{t-1} + b_f),\\
c_t(f_t, c_{t-1}, i_t, x_t, h_{t-1}) &= f_t \odot c_{t-1} + i_t \odot \tanh(\mathcal{W}_{xc}x_t + \mathcal{W}_{hc}h_{t-1} + b_c),\\
o_t(x_t, h_{t-1}, c_t) &= \sigma(\mathcal{W}_{xo}x_t + \mathcal{W}_{ho}h_{t-1} + \mathcal{W}_{co} \odot c_t + b_o),\\
h_t(o_t, c_t) &= o_t \odot \tanh(c_t),
\end{aligned} \tag{3.12}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ with $\sigma(x) = \frac{1}{1+e^x}$ and $\tanh$ are evaluated element-wise. The operation $\odot$ denotes the Hadamard product and the subscripts of the weight matrices $\mathcal{W} \in \mathbb{R}^{d^t_{\text{out}} \times d^t_{\text{in}}}$ describe the affiliation to the gates. For example, $\mathcal{W}_{xi}$ is the weight matrix to input x_t of gate i_t. This illustrates how the weights of the LSTMs are transferred to the weights of convolution LSTMs in the following.
We want to reformulate equation (3.12) by replacing all matrix-vector multiplications (i.e., the forward layer) by a convolution layer from Definition 3.1. This is inspired by [21], which has already provided the theoretic architecture of a convolutional LSTM layer with the approach on precipitation forecasting. Let $\text{Conv}(\,\cdot\,;\theta) : \mathbb{R}^{C_{\text{in}} \times H \times W} \to \mathbb{R}^{C_{\text{out}} \times H \times W}$ be a convolutional layer and $\mathcal{I} \in \mathbb{R}^{T \times C_{\text{in}} \times H \times W}$ a sequence of inputs ordered by the discrete time dimension $T \in \mathbb{N}$. A convolutional LSTM layer to an input sequence $\mathcal{I}_{t \le T}$ and $\mathcal{H}_0 = 0 \in \mathbb{R}^{C_{\text{in}} \times H \times W},\ \mathcal{C}_0 = 0 \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ (since at $t = 1$ we do not yet have any information about earlier steps in the sequence) is given by a system of equations,

$$\begin{aligned}
\hat{i}_t(\mathcal{I}_t, \mathcal{H}_{t-1}, \mathcal{C}_{t-1}) &= \sigma\Big(\text{Conv}(\mathcal{I}_t; \theta_{xi}) + \text{Conv}(\mathcal{H}_{t-1}; \theta_{hi}) + \mathcal{W}_{ci} \odot \mathcal{C}_{t-1}\Big),\\
\hat{f}_t(\mathcal{I}_t; \mathcal{H}_{t-1}, \mathcal{C}_{t-1}) &= \sigma\Big(\text{Conv}(\mathcal{I}_t; \theta_{xf}) + \text{Conv}(\mathcal{H}_{t-1}; \theta_{hf}) + \mathcal{W}_{cf} \odot \mathcal{C}_{t-1}\Big),\\
\mathcal{C}_t(\hat{f}_t, \mathcal{C}_{t-1}, \hat{i}_t, \mathcal{I}_t) &= \hat{f}_t \odot \mathcal{C}_{t-1} + \hat{i}_t \odot \tanh\Big(\text{Conv}(\mathcal{I}_t; \theta_{xc}) + \text{Conv}(\mathcal{H}_{t-1}; \theta_{hc})\Big),\\
\hat{o}_t(\mathcal{I}_t, \mathcal{H}_{t-1}, \mathcal{C}_t) &= \sigma\Big(\text{Conv}(\mathcal{I}_t; \theta_{xo}) + \text{Conv}(\mathcal{H}_{t-1}; \theta_{ho}) + \mathcal{W}_{co} \odot \mathcal{C}_t\Big),\\
\mathcal{H}_t(\hat{o}_t, \mathcal{C}_t) &= \hat{o}_t \odot \tanh(\mathcal{C}_t),
\end{aligned} \tag{3.13}$$

with $t \le T,\ t \in \mathbb{N}$ and $\mathcal{H} \in \mathbb{R}^{T \times C_{\text{out}} \times H \times W}$ the output of the convolutional LSTM layer as well as $\mathcal{W}_{ci}, \mathcal{W}_{cf} \in \mathbb{R}^{C_{\text{in}} \times H \times W}$, $\mathcal{W}_{co} \in \mathbb{R}^{C_{out} \times H \times W}$ in equation (3.12). The subscripts t indicate a cutout of the t-th element of sequence dimension T of the respective tensor.

Definition 3.3 (LSTM layer)**.** *Let $L, H, W, T, C_{in}, C_{out} \in \mathbb{N}$ as well as $\mathcal{W}_{ci}, \mathcal{W}_{cf} \in \mathbb{R}^{C_{in} \times H \times W}$, $\mathcal{W}_{co} \in \mathbb{R}^{C_{out} \times H \times W}$ and parameter vectors, specifying the convolutional layer as in Definition 3.2 for $L = 1$ from equation* (3.13)*,*

$$\theta_{xi} \in \mathbb{R}^{d_{xi}}, \theta_{hi} \in \mathbb{R}^{d_{hi}}, \theta_{xc} \in \mathbb{R}^{d_{xc}}, \theta_{hc} \in \mathbb{R}^{d_{hc}},$$

$$\theta_{xf} \in \mathbb{R}^{d_{xf}}, \theta_{hf} \in \mathbb{R}^{d_{hf}}, \theta_{xo} \in \mathbb{R}^{d_{xo}}, \theta_{ho} \in \mathbb{R}^{d_{ho}},$$

and described by the parameter vector $\theta \in \mathbb{R}^d$, with

$$d = d_{xi} + d_{hi} + d_{xc} + d_{hc} + d_{xf} + d_{hf} + d_{xo} + d_{ho} + 2 \cdot C_{in} \cdot H \cdot W + \cdot C_{out} \cdot H \cdot W.$$

Furthermore, let $\sigma : \mathbb{R} \to \mathbb{R}$ with $\sigma(x) = \frac{1}{1+e^x}$ and $\tanh$ evaluated element-wise. We call a function,

$$\textit{LSTM}(\,\cdot\,;\theta) : \mathbb{R}^{3 \times T \times C_{in} \times H \times W} \to \mathbb{R}^{2 \times T \times C_{out} \times H \times W},$$

a LSTM layer, if it satisfies the mapping rule given by the system of equations (3.13)*.*

For the forecasting of our gradient sequence, we use an encoder-decoder architecture (i.e., an "autoencoder") consisting of $2L,\ L \in \mathbb{N}$, LSTM layers,

$$\mathrm{LSTM}^l(\,\cdot\,;\theta^l) : \mathbb{R}^{3\times T\times C^l_{\text{in}}\times H\times W} \to \mathbb{R}^{2\times T\times C^l_{\text{out}}\times H\times W},$$

that satisfies the mapping rule given by the equation system (3.13) with $1 \le l \le 2L$. The encoding and decoding blocks of the autoencoder therefore have the same number of layers $L \in \mathbb{N}$. The autoencoder for an input sequence $\mathcal{I} \in \mathbb{R}^{T\times C_{in}\times H\times W}$ can be described by the following system of equations of the encoder block,

$$\begin{aligned}
\mathrm{LSTM}^1(\mathcal{I}_t, \mathcal{C}^1_{t-1}, \mathcal{H}^1_{t-1}; \theta^1) &= [\mathcal{C}^1_t, \mathcal{H}^1_t],\\
\mathrm{LSTM}^2(\mathcal{H}^1_t, \mathcal{C}^2_{t-1}, \mathcal{H}^2_{t-1}; \theta^2) &= [\mathcal{C}^2_t, \mathcal{H}^2_t],\\
&\vdots\\
\mathrm{LSTM}^{L-1}(\mathcal{H}^{L-2}_t, \mathcal{C}^{L-1}_{t-1}, \mathcal{H}^{L-1}_{t-1}; \theta^{L-1}) &= [\mathcal{C}^{L-1}_t, \mathcal{H}^{L-1}_t],\\
\mathrm{LSTM}^{L}(\mathcal{H}^{L-1}_t, \mathcal{C}^{L}_{t}, \mathcal{H}^{L}_{t-1}; \theta^{L}) &= [\mathcal{C}^{L}_t, \mathcal{H}^{L}_t],
\end{aligned} \tag{3.14}$$

with $1 \le t \le T_{\text{en}},\ t \in \mathbb{N}$. This is combined with the following decoder block by setting $\widetilde{\mathcal{O}}_0 = \mathcal{H}^L_{T_{\text{en}}}$ and $\mathcal{H}^{L+1}_0 = \mathcal{H}^1_{T_{\text{en}}}, \ldots, \mathcal{H}^{2L}_0 = \mathcal{H}^L_{T_{\text{en}}}$ and $\mathcal{C}^{L+1}_0 = \mathcal{C}^1_{T_{\text{en}}}, \ldots, \mathcal{C}^{2L}_0 = \mathcal{C}^L_{T_{\text{en}}}$,

$$\begin{aligned}
\mathrm{LSTM}^{L+1}(\widetilde{\mathcal{O}}_{t-1}, \mathcal{C}^{L+1}_{t-1}, \mathcal{H}^{L+1}_{t-1}; \theta^{L+1}) &= [\mathcal{C}^{L+1}_t, \mathcal{H}^{L+1}_t],\\
\mathrm{LSTM}^{L+2}(\mathcal{H}^{L+1}_t, \mathcal{C}^{L+2}_{t-1}, \mathcal{H}^{L+2}_{t-1}; \theta^{L+2}) &= [\mathcal{C}^{L+2}_t, \mathcal{H}^{L+2}_t],\\
&\vdots\\
\mathrm{LSTM}^{2L-1}(\mathcal{H}^{2L-2}_t, \mathcal{C}^{2L-1}_{t-1}, \mathcal{H}^{2L-1}_{t-1}; \theta^{2L-1}) &= [\mathcal{C}^{2L-1}_t, \mathcal{H}^{2L-1}_t],\\
\mathrm{LSTM}^{2L}(\mathcal{H}^{2L-1}_t, \mathcal{C}^{2L}_{t}, \mathcal{H}^{2L}_{t-1}; \theta^{2L}) &= [\mathcal{C}^{2L}_t, \widetilde{\mathcal{O}}_t],
\end{aligned} \tag{3.15}$$

with $1 \le t \le T_{\text{dec}},\ t \in \mathbb{N}$.

It should be mentioned that the input and output sequences do not have to be of the same length (in fact, in general $T_{\text{en}} \neq T_{\text{dec}}$). Furthermore, $C^{l-1}_{\text{out}} = C^l_{\text{in}}$ holds for individual LSTM layers $2 \le l \le 2L$ defined in equation (3.14) and (3.15). Especially, since the output sequence $\widetilde{\mathcal{O}}$ is also input of LSTM^{L+1}, it holds $C^{2L}_{\text{out}} = C^{L+1}_{\text{in}}$. In order to be able to select the dimensions of the output tensor $\widetilde{\mathcal{O}}$ completely independently of the hidden channels, we additionally apply a convolutional layer $\mathrm{Conv}(\,\cdot\,;\theta^{\text{final}}) : \mathbb{R}^{C^{2L}_{\text{out}}\times H\times W} \to \mathbb{R}^{C_{\text{final}}\times H\times W}$ with activation function $\sigma_{\text{final}} : \mathbb{R} \to \mathbb{R}$ to concatenate the hidden channel to an arbitrary number of output channels $C_{\text{final}} \in \mathbb{N}$ given by

$$\mathrm{Conv}(\widetilde{\mathcal{O}}_t; \theta^{\text{final}}) = \mathcal{O}_t \quad , 1 \le t \le T_{\text{dec}}. \tag{3.16}$$

Definition 3.4 (TLSTM architecture). *Let $L, H, W, T_{en}, T_{dec}, C_{final}, C^1_{in} \in \mathbb{N}$ as well as $C^l_{out} \in \mathbb{N}$, with $1 \le l \le 2L$ be given. Hence, the respective LSTM layers from the encoder defined in equation* (3.14) *and decoder in* (3.15) *block as well as the output layer in equation* (3.16) *can be described by the parameter vectors of the CNN and LSTM layers (see Definition 3.2 for $L = 1$ and Definition 3.3) $\theta^{final} \in \mathbb{R}^{d^f}, \quad \theta^l \in \mathbb{R}^{d^l}$ with $d^f \in \mathbb{N}, d^l \in \mathbb{N}$ for $1 \le l \le 2L$ and an activation function $\sigma_{final} : \mathbb{R} \to \mathbb{R}$ of the output layer. These parameter vectors as well as the weight tensors $\mathcal{W}^l_{ci}, \mathcal{W}^l_{cf} \in \mathbb{R}^{C_{in}\times H\times W}$ and $\mathcal{W}^l_{co} \in \mathbb{R}^{C_{out}\times H\times W}$ for $l \le 2L$ can in turn be described collectively by the parameter vector $\theta \in \mathbb{R}^d$, where*

$$d = d_f + \sum_{l=1}^{2L} d^l.$$

We call a NN as described in equations (3.14)–(3.16) *Convolutional Topology Long Short-Term Memory (TLSTM) and characterise it by*

$$\mathcal{N}_{LSTM}(\,\cdot\,;\theta) : \mathbb{R}^{T_{en}\times C_{in}\times H\times W} \to \mathbb{R}^{T_{dec}\times C_{final}\times H\times W}.$$

The difference between a TLSTM and an LSTM is therefore the structure of the input tensor $\mathbb{R}^{T_{\text{en}}\times C_{in}\times H\times W}$ of a TLSTM instead of $\mathbb{R}^{T_{\text{en}}\times C_{\text{in}}\times H}$ and the internal calculation carried out with convolutional layers instead of standard multiplications.

Example 3.3. *For the experiments in Section 4.2, the underlying $L = 4$ layer TLSTM with $C_{in} = 3$ input channels, $C^l_{out} = 9, l \leq 4$ hidden channels, $C_{final} = 1$ output channel, kernel size of all included convolutional layers $H_K = W_K = 3$ and sequence lengths $T_{en} = 5$, $T_{dec} = 10$ with trained weights described by $\theta \in \mathbb{R}^{509266}$ are given as*

$$\mathcal{N}_{LSTM}\,\cdot\,;\theta) : \mathbb{R}^{5\times 3\times 201\times 101} \to \mathbb{R}^{10\times 1\times 201\times 101}, \tag{3.17}$$

with activation function $\sigma^{final}(x) = \min\{\max\{x, 0\}1\}$. The autoencoder structure for a 8 layer LSTM described in equations (3.14), (3.15) *and* (3.16) *for* (3.17) *is visualised in Figure 3.*

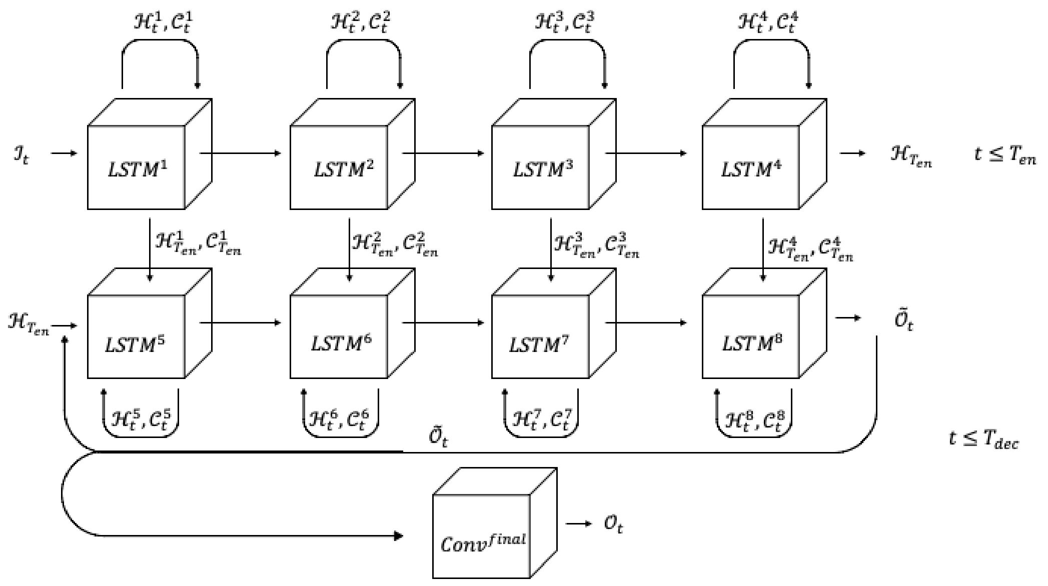

$\mathcal{H}^1_t, \mathcal{C}^1_t$
$\mathcal{H}^2_t, \mathcal{C}^2_t$
$\mathcal{H}^3_t, \mathcal{C}^3_t$
$\mathcal{H}^4_t, \mathcal{C}^4_t$
$\mathcal{I}_t$
$LSTM^1$
$LSTM^2$
$LSTM^3$
$LSTM^4$
$\mathcal{H}_{T_{en}}$
$t \leq T_{en}$
$\mathcal{H}^1_{T_{en}}, \mathcal{C}^1_{T_{en}}$
$\mathcal{H}^2_{T_{en}}, \mathcal{C}^2_{T_{en}}$
$\mathcal{H}^3_{T_{en}}, \mathcal{C}^3_{T_{en}}$
$\mathcal{H}^4_{T_{en}}, \mathcal{C}^4_{T_{en}}$
$\mathcal{H}_{T_{en}}$
$LSTM^5$
$LSTM^6$
$LSTM^7$
$LSTM^8$
$\tilde{O}_t$
$\mathcal{H}^5_t, \mathcal{C}^5_t$
$\mathcal{H}^6_t, \mathcal{C}^6_t$
$\tilde{O}_t$
$\mathcal{H}^7_t, \mathcal{C}^7_t$
$\mathcal{H}^8_t, \mathcal{C}^8_t$
$t \leq T_{dec}$
$Conv^{final}$
O_t

Figure 3: Autoencoder architecture $\mathcal{N}_{\text{LSTM}}$ of Example 3.3.

3.4 Data preparation

As in the case of the integration of the $\mathcal{N}_{\text{CNN}}$ proposed in Section 3.1, we want to replace lines 5 and 6 in Algorithm 4 with the approximation of the $\mathcal{N}_{\text{LSTM}}$ from Definition 3.4. In case of a TLSTM, the evaluations of φ and u have to be transformed from the graph structure of the finite element simulation into an appropriate tensor format. This in principle is analogous to the composition $\Phi_{C_{\text{in}}} \circ p$ in equation (3.9). The only difference is that now this is performed on a sequence of evaluations φ_{m_n} and u_{m_n} of length $T_{\text{en}} \in \mathbb{N}$. As the subscripts suggest, such a sequence does not necessarily have to be evaluated on a fixed mesh $\mathcal{T}_m$, it may extend over a sequence of meshes $\mathcal{T}_m$. However, since we use polynomial interpolations

$$\begin{aligned} p_T &: \mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{in}}\times T} \to \mathbb{R}^{H\cdot W\times C_{\text{in}}\times T},\\ q_T &: \mathbb{R}^{H\cdot W\times C_{\text{out}}\times T} \to \mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{out}}\times T} \end{aligned}$$

to transfer the sequence $\varphi_{n-T},\dots,\varphi_n$ and $u_{n-T+1},\dots,u_{n+1}$ onto some reference mesh $\mathcal{T}_{\text{const}} = (V(\mathcal{T}_{\text{const}}), E(\mathcal{T}_{\text{const}}))$.

We intend to process T_{en} feature matrices of the form of equation (3.7). Hence, we define transformations

$$\begin{aligned} \Phi_{C_{\text{in}},T} &: \mathbb{R}^{H\cdot W\times C_{\text{in}}\times T} \to \mathbb{R}^{T\times C_{\text{in}}\times H\times W},\\ \Phi_{C_{\text{out}},T} &: \mathbb{R}^{T\times C_{\text{out}}\times H\times W} \to \mathbb{R}^{H\cdot W\times C_{\text{out}}\times T}. \end{aligned} \tag{3.18}$$

Thus, the approximation of a gradient step in Algorithm 4 by a TLSTM can be understood as a concatenation of the form

$$\begin{aligned} &\mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{in}}\times T_{\text{en}}} \xrightarrow{p_T} \mathbb{R}^{H\cdot W\times C_{\text{in}}\times T_{\text{en}}} \xrightarrow{\Phi_{C_{\text{in}},T_{\text{en}}}} \mathbb{R}^{T\times C_{\text{in}}\times H\times W} \xrightarrow{\mathcal{N}_{\text{CNN}}}\\ &\xrightarrow{\mathcal{N}_{\text{CNN}}} \mathbb{R}^{T\times C_{\text{out}}\times H\times W} \xrightarrow{\Phi_{C_{\text{out}},T_{\text{dec}}}} \mathbb{R}^{H\cdot W\times C_{\text{out}}\times T_{\text{dec}}} \xrightarrow{q_T} \mathbb{R}^{|V(\mathcal{T}_m)|\times C_{\text{out}}\times T_{\text{dec}}} \end{aligned} \tag{3.19}$$

Example 3.4 (The TLSTM)**.** *The NN given by the coupling of functions from equation* (3.19)*, where $\mathcal{N}_{LSTM}$ as given in Example 3.3, can be described by*

$$\mathcal{N}_{LSTM}(\,\cdot\,;\theta) : \mathbb{R}^{|V(\mathcal{T}_m)|\times 3\times 5} \to \mathbb{R}^{|V(\mathcal{T}_m)|\times 10} \tag{3.20}$$

and

$$\begin{bmatrix} \varphi_{n-5} & u^x_{n-5+1} & u^y_{n-5+1}\\ & \vdots & \\ \varphi_n & u^x_{n+1} & u^y_{n+1} \end{bmatrix} \mapsto \begin{bmatrix} \varphi_{n+1}\\ \vdots\\ \varphi_{n+10} \end{bmatrix}$$

for $\varphi_n \in \mathbb{R}^{|V(\mathcal{T}_m)|}$ and $u_n = (u^x_n, u^y_n),\ u^x_n, u^y_n \in \mathbb{R}^{|V(\mathcal{T}_m)|}$ defined on mesh $\mathcal{T}_m$.

The TLSTM from Example 3.4 can directly be integrated into Algorithm 4 like before with Algorithm 1. In fact, since in Algorithm 4 only the most recent gradient step is relevant, in practice we restrict the inverse mapping on the last element of the predicted sequences to save calculation time. The only difference is that the sequences $\varphi_{n-T},\dots,\varphi_n$ and $u_{n-T+1},\dots,u_{n+1}$ have to be stored in a list. We chose $c_1 = 125$ and $c_m = 50$ for all $2 \le m \in \mathbb{N}$. This procedure is described in Algorithm 3. As in case of the TCNN, we are able to include the extra sample dimension S to approximate gradient steps from multiple problems at ones.

4 Results

This section is devoted to numerical results of the two previously described neural network architectures. The implementations were done with the open source packages `PyTorch` [19] for the NN part and `FEniCS` [2] for the FE simulations[2]. We first illustrate the performance of the TCNN in Section 4.1 with a deterministic bridge example compared to a classical optimisation. The important observation is that with the TCNN the optimisation can be carried out with far fewer optimisation steps while still leading to the reference topologies from [10]. Similar results can be observed for the risk-averse stochastic optimisation. In Section 4.2, numerical experiments of the TLSTM architecture are presented. It turns out that the performance is comparable to the TCNN architecture and the optimisation seems to be more robust with respect to the data realisations.

[2]see introduction for the link to the code repository

Algorithm 3: Deterministic optimisation algorithm with TLSTM approximated gradient step.

```
Input: mesh $\mathcal{T}_0$, $\mathcal{T}_{\text{const}}$, initial values $\varphi_0$, sequence $c_m \in \mathbb{N}$ and $j = 1, T \in \mathbb{N}$ and list $L = [\varphi_0]$
for $m = 0, 1, \dots$ until converged do
    for $n = 0, 1, \dots$ until converged do
        solve state equation on mesh $\mathcal{T}_m \Rightarrow u_{n+1}$
        add $u_{n+1}$ to $L$
        if $j = c_m$ then
            interpolate $\varphi_{n-T}, \dots, \varphi_n$ and $u_{n-T1}, \dots, u_{n+1}$ on to $\mathcal{T}_{\text{const}}$
            project $\varphi_{n-T}, \dots, \varphi_n$ and $u_{n-T+1}, \dots, u_{n+1}$ on to $\mathcal{T}_{\text{const}}$ to $\mathbb{R}^{T \times d+1 \times H \times W}$
            evaluate $\mathcal{N}^T_{\text{LSTM}}(\varphi_{n-T}, \dots, \varphi_n, u_{n-T+1}, \dots, u_{n+1}; \theta)$ like in equation (3.19)
              $\Rightarrow \varphi_n \cdots \varphi_{n+T}$
            project $\varphi_{n+T}$ to $\mathcal{T}_{\text{const}}$
            interpolate $\varphi_{n+T}$ onto $\mathcal{T}_m$
            declare list and add $\varphi_{n+T} \Rightarrow L = [\varphi_{n+T}]$
            $j = 1$
        else
            solve adjoint equation on mesh $\mathcal{T}_m \Rightarrow p_{n+1}$
            solve gradient equation on mesh $\mathcal{T}_m \Rightarrow \varphi^*_{n+1}$
            project $\varphi^*_{n+1}$ to $[0, 1] \Rightarrow \varphi_{n+1}$
            add $\varphi_{n+1}$ to $L$
            $j = j + 1$
        end
    end
    declare list $L$
    $j = 1$
    adapt mesh according to Section B $\Rightarrow \mathcal{T}_{m+1}$
end
```

4.1 TCNN examples

Before we can use the TCNN architecure for the optimisation in Algorithm 1, we have to train it on data which describes the system response of equation (2.6). Note that it would not be useful to let the $\mathcal{N}_{\text{CNN}}$ learn the gradient steps of a fixed setting since different settings of the bridge problem from Section A.1 should efficiently be tackled. In considering the stochastic setting of the problem as defined in Section 2.2, the TCNN is trained to learn the gradient steps φ for a random $g : \Omega \to \mathbb{R}^d$.

4.1.1 Sampling the data

To train the architecture, appropriate training data has to be generated. In order to achieve this, we chose the same setting for g as in Section A.2. Using the optimiser in Algorithm 4, we can generate $S \in \mathbb{N}$ different sample paths of gradient steps $\varphi_n(g)$ and solutions of the state equation $u_n(g)$ by generating S samples of g. In this procedure, we store every $k \in \mathbb{N}$ iteration step of φ_n and u_n in order to approximate k gradient steps at once. More precisely, we store $\lfloor \frac{N_{\max}}{k} \rfloor - 1$ tuples

$$\left(\begin{bmatrix} \varphi_n & u^x_{n+1} & u^y_{n+1} \end{bmatrix}, \begin{bmatrix} \varphi_{n+k} \end{bmatrix} \right), \tag{4.1}$$

with $0 \leq n \leq N_{\text{max}} - k$, where $N_{\text{max}} \in \mathbb{N}$ is fixed in advance, representing the maximum number of iterations of an optimisation. For the training of the models in the following experiments, we have chosen $N_{\text{max}} = 500$ since the topologies have mostly converged after this number of iterations. The overall number of $S(\lfloor \frac{N_{\text{max}}}{k} \rfloor - 1)$ tuples are merged into an unsorted data set $\mathcal{D}_{\text{CNN}}$.

4.1.2 TCNN predictions

The following experiment validates that the performance of Algorithm 4 can be replicated or (desirably) improved by including a CNN as described in Algorithm 1. As a first test, we illustrate that the proposed new architecture is actually capable of predicting the gradients of the optimisation procedure.

Figure 4 shows the evaluations of the model equation (3.10) after determining θ within the training of the NN on the data set $\mathcal{D}_{CNN}$. Here, the prediction $\mathcal{N}^k_{\text{CNN}}(\varphi_n, u_{n+1}; \theta)$ and the actual gradient step φ_{n+k} generated by Algorithm 4 are compared for different loads sampled from a truncated normally distributed g. Since the predictions of $\mathcal{N}^5_{\text{CNN}}$ are hardly distinguishable from the reference fields, we have also trained equation (3.10) to predict larger time steps (for 25 and 100 iteration steps at once). However, these NNs have proved to be less reliable in practice as prediction quality decreases. An illustrative selection of some predictions is provided in Figures 16 and 17 in Appendix C.

4.1.3 Deterministic bridge optimisation

In these experiments we compare the performance of Algorithm 1 with that of Algorithm 4 in the setting of Section A.1. As NN in Algorithm 1 we use equation (3.10) from Example 3.2. For $c_m \in \mathbb{N}$ in Algorithm 1 we chose $c_m = 55$ for all $m \in \mathbb{N}$. In order to train equation (3.10), data of the form of equation (4.1) from Section 4.1.1 is used. We expect that the best (reference) results in the setting from experiment A.1 can be obtained, since the distribution of the training data is a truncated normal distribution around this expected value and we thereby have an accumulation of training points around the load $g = (0, -5000)^T$. As a convergence criterion, we used the convergence criterion of the mesh refinement of equation (B.1) on a maximally fine mesh with $|V(\mathcal{T}_m)| \leq 15000$. A sub-sequence generated by Algorithm 1 is shown in Figure 18a in comparison to that of Section A.1 in Figure 18b in Appendix C. There it can be observed that the classical and the CNN assisted optimisation results look basically identically with the CNN converging faster.

The metrics for evaluating our algorithms have also improved through the application of the CNN as can be observed in Figure 5. For better comparability, we ran both algorithms ten times and averaged all metrics. To be more precise, this is only the average of the calculation time, as the remaining metrics are deterministic and therefore always the same. It is easy to see how the metrics diverge with the first application of the CNN at iteration step 55, especially by the computation time required per iteration. The most significant indicator is the evaluation of $J^{\varepsilon}(\varphi_n)$ per iteration of equation (2.3) at the top right of Figure 5. The graph for Algorithm 1 reaches a constant lower level than the one of Algorithm 4 after about 250 iterations and thus fulfils the convergence criterion earlier. Accordingly, the step size criterion for τ_n applies earlier by using the CNN, which further accelerates convergence. An interesting insight is provided by the calculation time, which shows that the actual time required per iteration step is more or less the same, except for the iteration steps in which equation (3.10) is applied. This is indicated by the upward outliers in the computation time series. This additional computation cost can be explained by the application of the mesh projection of equation (3.9), which is an aspect that requires further improvements. Nevertheless, in total we achieve a shorter total run-time due to the faster convergence of Algorithm 1.

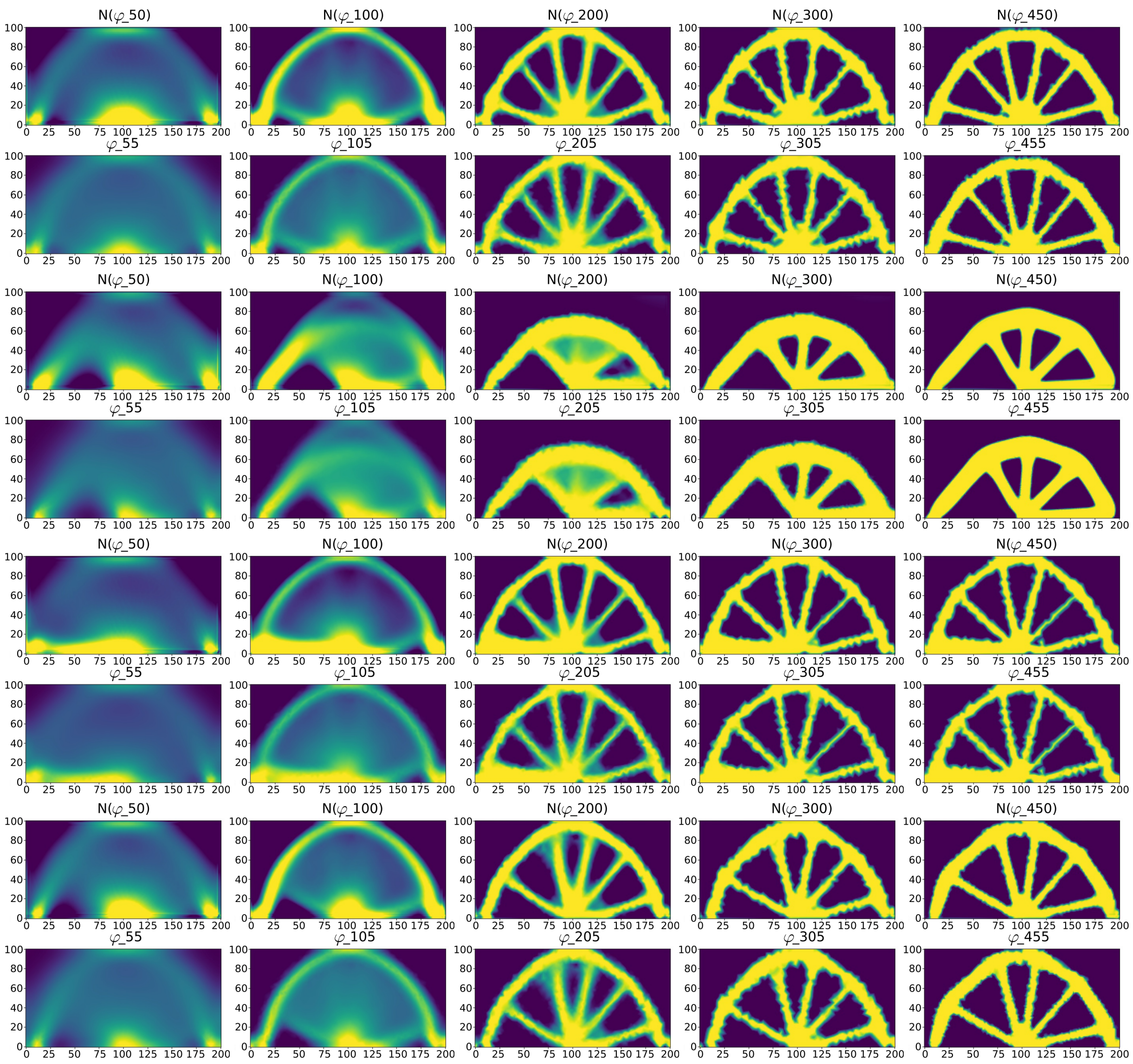


Figure 4: $\mathcal{N}^5_{CNN}(\varphi_n, u_{n+1}; \theta)$ (top row) in comparison with φ_{n+5} (bottom row).

A detailed list of the run-times and target value metrics for the functional $J^\varepsilon(\varphi_n)$ is provided in Table 1. The evaluation of $J^\varepsilon(\varphi_{n_{\text{final}}})$ denotes the value of the functional for the topology $\varphi_{n_{\text{final}}}$ converged after $n_{\text{final}} \in \mathbb{N}$ iteration steps. The compliance is the value that is actually minimised in terms of the functional $J^\varepsilon(\varphi_{n_{\text{final}}})$. Algorithm 1 requires less computing time than the reference procedure after extending it with the CNN architecture from equation (3.10).

4.1.4 Stochastic bridge optimisation

Algorithm 5 can easily be extended by the TCNN of equation (3.10) in order to improve the efficiency for topology optimisation under uncertainties. The corresponding procedure is shown in Algorithm 2 where we chose $c_m = 55$ for all $m \in \mathbb{N}$. Note that the predictions of the different realisations of $\varphi_n(\omega_i), \omega_i \in \Omega$ for $i = 1, \dots, S \in \mathbb{N}$ (where an evaluation $\varphi_n(\omega_i)$ is to be interpreted as transformation of an evaluation from $g(\omega_i)$) are actually not executed within a loop but in parallel (lines 8-12 of Algorithm 2). This is possible because NNs are generally able to process batches of data in parallel. We have also implemented parallelisation for Algorithm 5, which is limited by the number of

method	applied load g	converges after n_{final}	evaluation of $J^{\varepsilon}(\varphi_{n_{\text{final}}})$	compliance $\int_{\Gamma_g} g \cdot u(\varphi)\,\mathrm{d}s$
Algorithm 4	$(0,-5000)^T$	538	1475.39	1173.22
Algorithm 1	$(0,-5000)^T$	**441**	1206.49	**1148.66**

Table 1: Comparison of metrics between Algorithms 4 and 1.

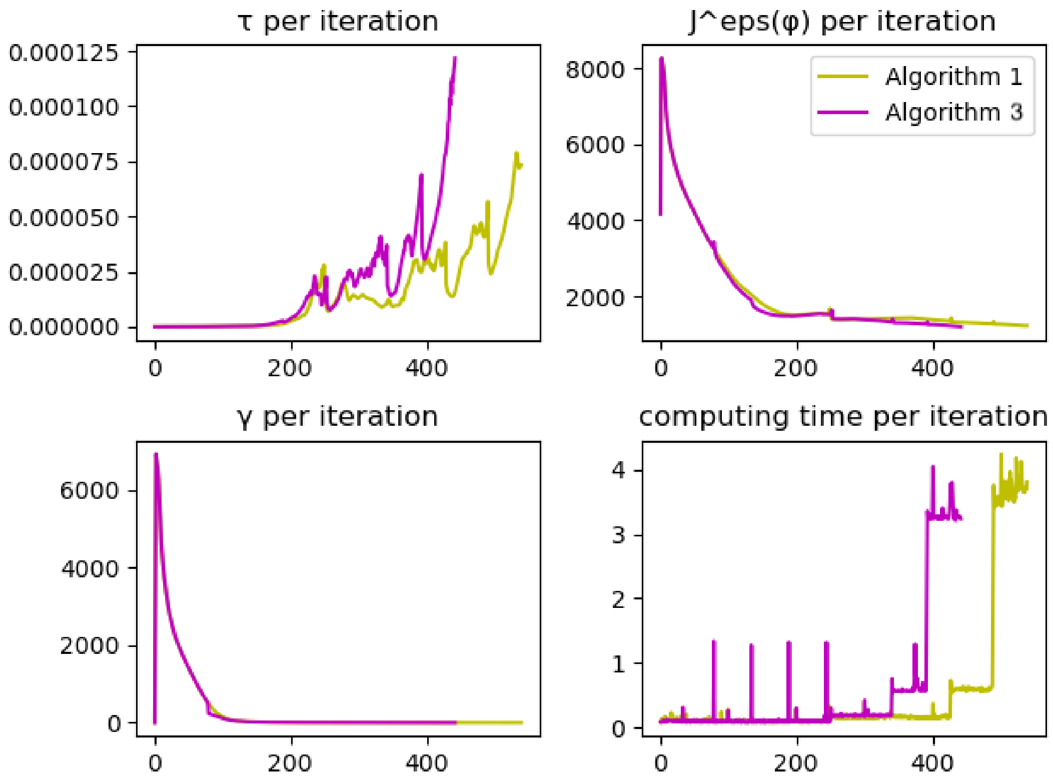
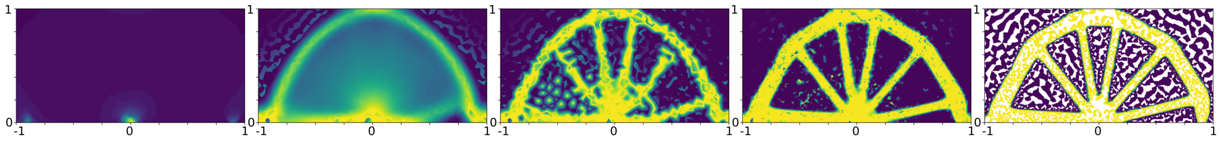


Figure 5: Comparison of metrics between Algorithms 4 and 1.

processor cores of the actual compute cluster. We want to compare the performance of Algorithm 5 and Algorithm 2 using the same setting as in Section A.2. To ensure comparable results despite stochastic parameters, we set the random seed to 42 before running both algorithms. The resulting sub-sequences of φ_n are compared side by side in Figure 6. Although the topology converges after fewer iterations with Algorithm 5, one can see that the topology resulting from Algorithm 2 has a more stable shape since the topology does not lose material to the unnecessary extra spoke. This is confirmed by the metrics in Table 3 where one can see that Algorithm 2 achieved a lower compliance after fewer iteration steps. Additionally, the optimisation of Algorithm 2 is stopped after 500 iterations to show that it achieves a better result after less time as shown in Figure 8. A notable observation is that the times of applying equation (3.10) in Algorithm 2 can be identified by the spikes in the computation time of the iterations. It can be seen that despite the additional time that the transformation in equation (3.9) requires, the calculation of a stochastic gradient step using equation (3.10) is generally faster. This is due to the dynamic parallelisation that `PyTorch` provides when processing batches (in our case the approximation of multiple evaluation from $\varphi_n(\omega_i)$ with NNs). However, the amount of evaluations of $\varphi_n(\omega_i)$ that Algorithm 5 can process at once is limited by the number of available processors. Since the calculation time for the evaluation of an optimisation step $\varphi_{n+1}(\omega_i)$ increases with finer meshes, the evaluation of the approximation of all gradient steps $\varphi_{n+1}(\omega_1), \dots, \varphi_{n+1}(\omega_S)$ at once results in a processing time advantage for the NN. It is to be expected that this time saving increases with the number of examples S.


(a) Sequence $\varphi_1, \varphi_{100}, \varphi_{200}, \varphi_{400}, \varphi_{517}$ from Algorithm 2.

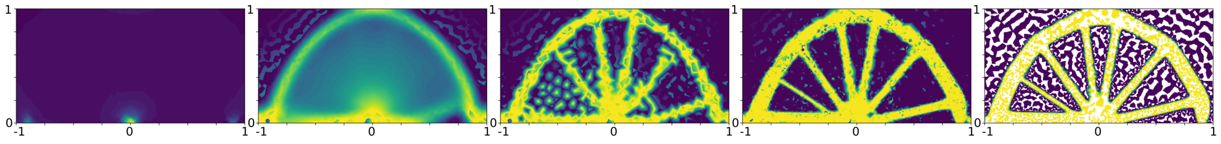

(b) Sequence $\varphi_1, \varphi_{100}, \varphi_{200}, \varphi_{400}, \varphi_{490}$ from Algorithm 5.

Figure 6: Classical risk-averse stochastic optimisation (top) and $\mathcal{N}_{\text{CNN}}$ accelerated (bottom).

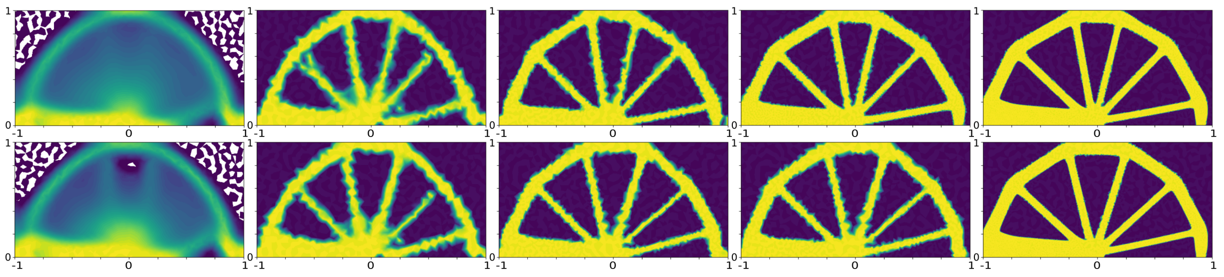
As mentioned at the beginning of the experiment, we expect to achieve good results close to the mean of $g = (0, -5000)^T$. In order to get a more general view of the quality of Algorithm 1, we have compiled a selection of extreme cases for the distribution of g (e.g., evaluations from g that deviates strongly from $(0, -5000)^T$) in Figure 7 and Table 2. The figure shows the sequence of φ_n in hundreds of steps as well as the final distribution of material ($\varphi_{100}, \varphi_{200}, \dots, \varphi_{n_{\text{final}}}$) for the specific loads g. Table 2 indicates a noticeable saving in calculation time, but there is no guaranteed improvement in the results. In particular, when the topology “collapses” (i.e., the NN cannot generalise to the input data with strong deviations from the training data), the application of the CNN leads to worse results. Nevertheless, it can be seen that the NN extension gives the algorithm a greater robustness against porous fragments (see Figure 7b) in the optimisation of the topology and thus a higher stability against collapsing of the topology in the optimisation can be assumed. Finally, a critical aspect to be mentioned is the step size c_m. The time at which equation (3.10) is applied and which is controlled by $c_m \in \mathbb{N}$ has a crucial impact on the viability or “compatibility” between the state of the optimisation procedure and the CNN. In some cases, a c_m that is too small or too large can lead to the collapse of the topology, i.e., the topology deteriorates and does not recover. For the reliable use of Algorithm 1, a method for controlling c_m would have to be devised.

method	applied load g	converges after n_{final}	evaluation of $J^{\varepsilon}(\varphi_{n_{\text{final}}})$	compliance $\int_{\Gamma_g} g \cdot u(\varphi)\,\mathrm{d}s$
Algorithm 4	$(0, -5000)^T$	538	1475.39	1173.22
Algorithm 1	$(0, -5000)^T$	441	1206.49	**1148.66**
Algorithm 4 (see Fig. 7a)	$(2632.16, -4251.09)^T$	473	1487.39	1416.46
Algorithm 1 (see Fig. 7a)	$(2632.16, -4251.09)^T$	517	1475.05	**1405.39**
Algorithm 4 (see Fig. 7b)	$(-733.23, -4945.95)^T$	**457**	1115.66	1062.41
Algorithm 1 (see Fig. 7b)	$(-733.23, -4945.95)^T$	497	1049.91	**1042.58**
Algorithm 4 (see Fig. 7c)	$(-1099.92, -4877.25)^T$	470	1042.18	**992.28**
Algorithm 1 (see Fig. 7c)	$(-1099.92, -4877.25)^T$	447	1048.44	998.34

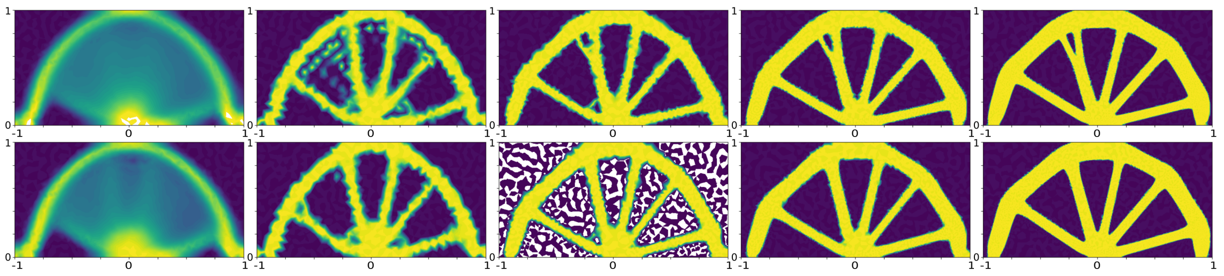
Table 2: Comparison of metrics between Algorithm 4 and 1.

4.2 TLSTM examples

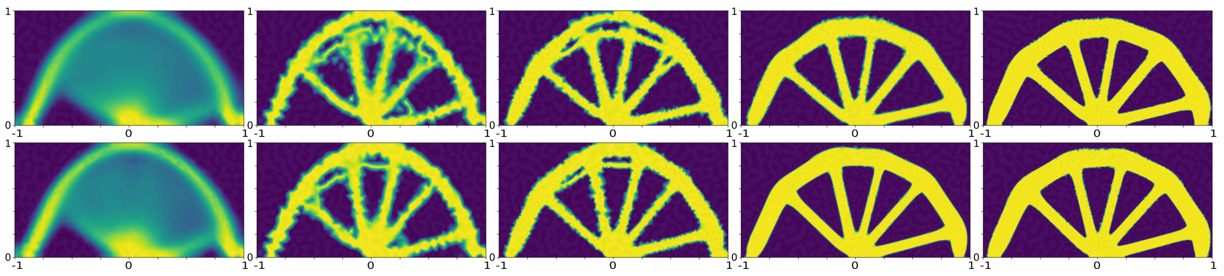
As in Section 4.1.3, randomly generated training data should be used in the following experiment with the derived LSTM transformation of equation (3.19). The data tuples consist of input and output from $\mathcal{N}_{\text{LSTM}}$ according to Example 3.4.



(a) $g = (2632.16, -4251.09)^\intercal$



(b) $g = (-733.23, -4945.95)^\intercal$



(c) $g = (-1099.92, -4877.25)^\intercal$

Figure 7: Comparison of metrics between Algorithms 4 (top row) and 1 (bottom row) for different loads g.

4.2.1 Sampling the data

Again, we assume an expected load $g = (0, -5000)^T$ and a random rotation characterised by a truncated normal distribution with bounds $[-\frac{\pi}{2}, \frac{\pi}{2}]$, standard deviation 0.3 and mean 0. Using the optimiser in Algorithm 4, $S \in \mathbb{N}$ sample paths of gradient steps $\varphi(g)$ and solutions of the state equation $u(g)$ are generated by drawing S realisations of g. In contrast to Section 4.1.1, this time we do not only store every $T \in \mathbb{N}$ iteration step of φ_n and u_n but instead store all iteration steps of the optimisation of Algorithms 4. Afterwards these are merged into disjoint subsets, each consisting of a sequence of T iteration steps. Thus, the feature or the sequence $\varphi_{n+1}, \dots, \varphi_{n+T}$ is the label for the assembled sequence from $\varphi_{n-T}, \dots, \varphi_n$ and $u_{n-T+1}, \dots, u_{n+1}$. More precisely, with

$$\left(\begin{bmatrix} \varphi_{n-T} & u^x_{n-T+1} & u^y_{n-T+1} \\ & \vdots & \\ \varphi_n & u^x_{n+1} & u^y_{n+1} \end{bmatrix}, \begin{bmatrix} \varphi_{n+1} \\ \vdots \\ \varphi_{n+T} \end{bmatrix} \right),$$

for $0 \leq n \leq N - (T-1)$, a total of $S(\lfloor \frac{N}{k} \rfloor - 1)$ tuples are stored in an unsorted dataset $\mathcal{D}_{\text{LSTM}}$.

method	converges after n_{final}	evaluation of $J_0^{\varepsilon}(\varphi_{n_{\text{final}}})$	compliance $\mathbb{E}\left[\int_{\Gamma_g} g \cdot u(\varphi_{n_{\text{final}}})\,\mathrm{d}s\right]$	samples
Algorithm 5	490	765.85	729.20	224
Algorithm 2	517	744.11	708.61	224
Algorithm 2	500	760.26	723.24	224

Table 3: Comparison of metrics between Algorithm 5 and 2.

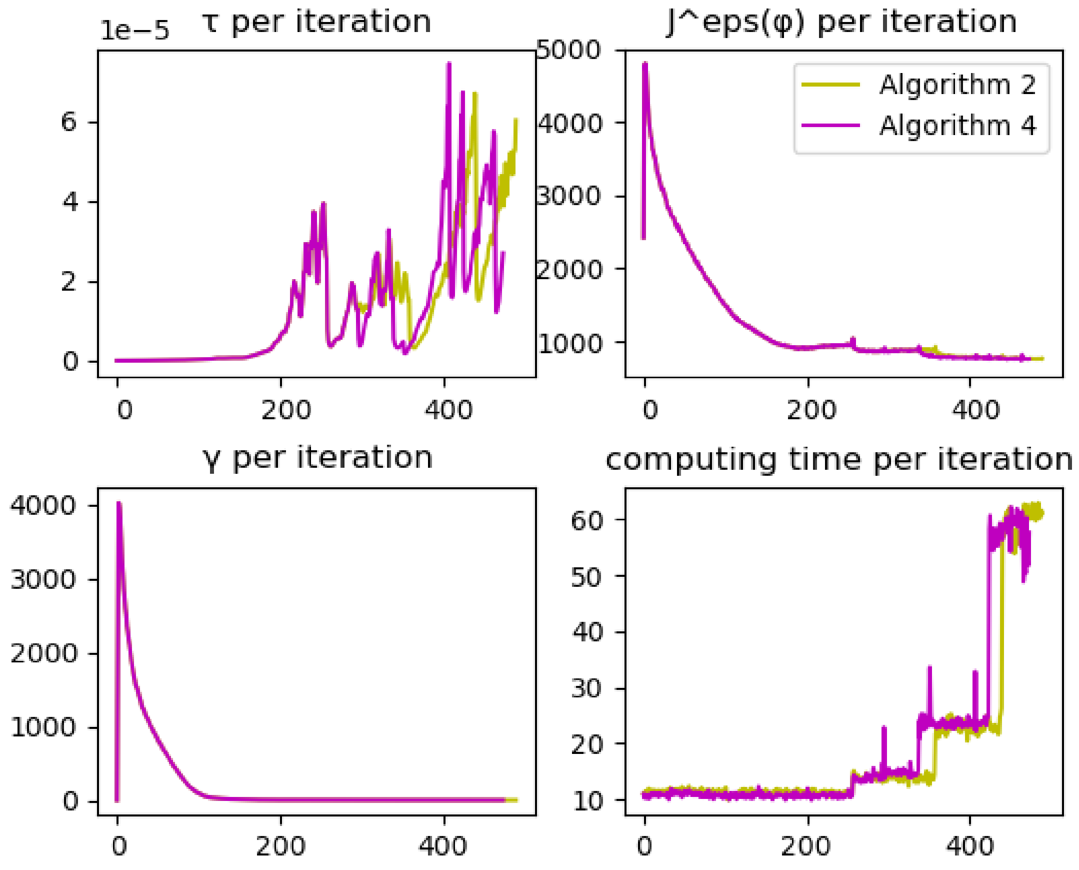


Figure 8: Comparison of metrics between Algorithm 5 and 2.

4.3 TLSTM predictions

After training the TLSTM from equation (3.17) with the data set $\mathcal{D}_{\text{LSTM}}$ generated in Section 4.2.1, we want to investigate its predictive ability of $\mathcal{N}_{\text{LSTM}}^{T}(\varphi_n) := \mathcal{N}_{\text{LSTM}}^{T}(\varphi_{n-T}, \dots, \varphi_n, u_{n-T+1}, \dots, u_{n+1}; \theta)$ compared to the real sequence $(\varphi_{n+T})_n$. For this purpose we have visualised both sub-sequences for $T = 10$ in Figure 9 using the iteration sequence generated for a load $g = (0, -5000)$. The distorted topology in the first forecasts is striking, this can be attributed to the comparatively low weighting of training data in which the distribution of the material is constant $\varphi_n(x) = 0,5$ for all $x \in D$ or almost constant. This forces us to choose a correspondingly high $c_m \in \mathbb{N}$ in Algorithm 3. Furthermore, it can be seen that especially in early phases of the partial sequence in which the change $\|\varphi_n - \varphi_{n+1}\|$ is very high, $\mathcal{N}_{\text{LSTM}}^{10}(\varphi_n)$ provides a better forecast from an visual perspective, i.e., the topology $\mathcal{N}_{\text{LSTM}}^{10}(\varphi_n)$ has already converged further than the target image φ_{n+10}. Since the topologies on the finer meshes no longer show any major visual changes and therefore the differences in the predictions are no longer recognisable, we have decided not to present them at this point.

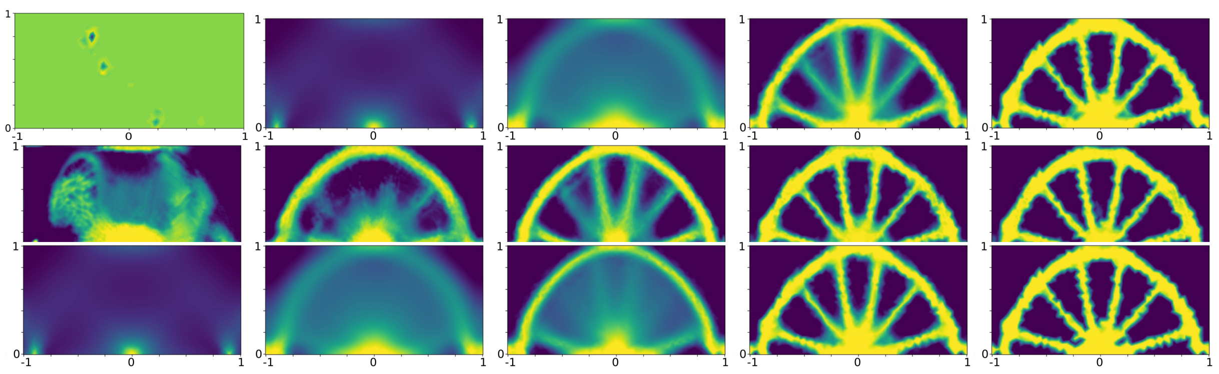


Figure 9: Input φ (top row), $\mathcal{N}^{10}_{\text{LSTM}}(\varphi_n)$ (center row) in comparison with φ_{n+10} (bottom row).

The architecture of the TLSTM allows the length of the input sequence as well as the output sequence to be chosen independently of the training data. The expected consequence is a decrease in prediction quality. Despite this, Figure 10 depicts the prediction results of equation (3.17) with unchanged input sequence ($T_{en} = 5$) and output sequence of length 40. Since a shorter output sequence ($T_{des} = 10$) is used to train equation (3.17), the results of the longer output sequence indicate that N_{LSTM} has indeed learned to predict a gradient step for the given setting and that the training data from Section 4.2.1 describes the problem correctly.

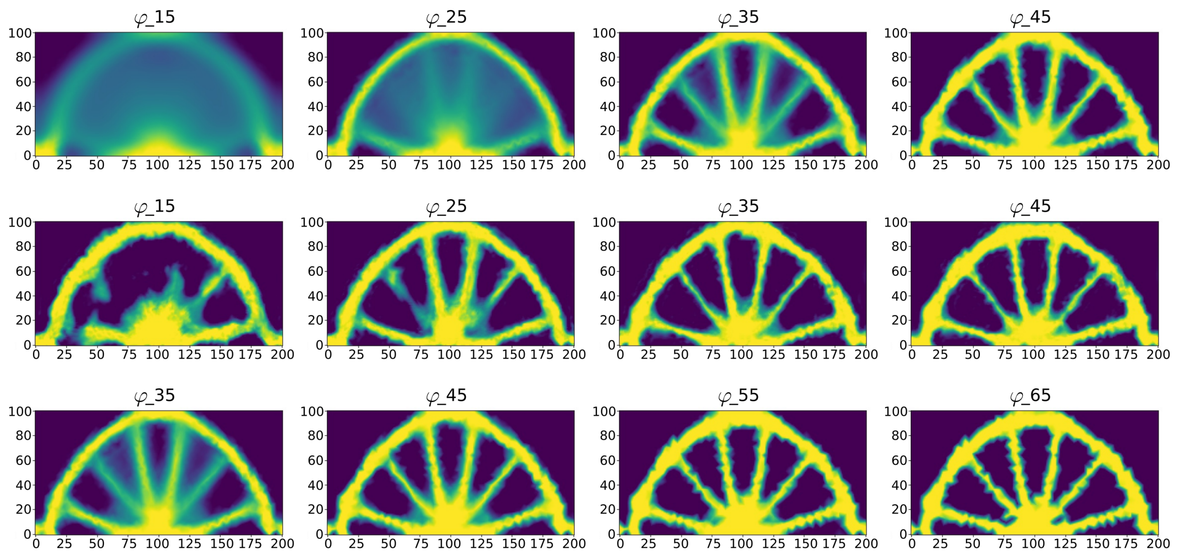


Figure 10: Input φ (top row), $\mathcal{N}^{20}_{\text{LSTM}}(\varphi_n)$ (center row) in comparison with φ_{n+20} (bottom row).

Analogous to Section 3.1.1, the intention behind the construction of $\mathcal{N}_{\text{LSTM}}$ is to replace the gradient step in reference Algorithm 4 by Example 3.3. Algorithm 3 describes the integration of the LSTM prediction. When evaluating $\mathcal{N}^{T}_{\text{LSTM}}$, only the last iteration step φ_{n+T} of the predicted sequence is projected back to the current mesh $\mathcal{T}_m$ in order to save computational resources. We examine the performance of Algorithm 3 in the following deterministic experiment. As for the TCNN above, the beneficial performance of a single gradient prediction transfers to the stochastic setting since it consists of a Monte Carlo estimator with $N \in \mathbb{N}$ samples in each step. It is hence not necessary to examine this in more detail.

4.4 Deterministic bridge optimisation

The motivation for the design of the TLSTM architecture was that the information contained in the time series of φ_n and u_n could ideally lead to an improvement in the forecast capabilities of the NN. This can be investigated as in Section 4.1.4 by calculating the optimal topology for different loading scenarios for g by Algorithm 3. Again, all results are based on the tenfold averaged performance of the algorithms for each load g. Figure 11 shows the results in the setting similar to Section A.1 and compares the respective metrics. Analogous to Section 4.1.3 the sequence φ_n of the optimisation by Algorithm 3 is also more resilient to porous fragments in the structures than the reference optimisation procedure. In general, the pictures of φ_n hardly differ between Algorithms 1 and 3. Hence, apart from Figure 11, no further visualisations are presented. It should also be noted that the optimisation by Algorithm 3 is much less stable than the optimisation using $\mathcal{N}_{\text{CNN}}$ from equation (3.11). This becomes apparent when the structure collapses which was the case in each of our test runs if $c_m \in \mathbb{N}$ was chosen too small in Algorithm 3. Furthermore, it also could be observed that the convergence criterion of equation (B.1) was not reached after applying $\mathcal{N}_{\text{LSTM}}$ because φ_n diverged too much from the actual minimum on $\mathcal{T}_m$. In conclusion, the stability of Algorithm 3 is even more dependent on c_m than it is with Algorithm 1, which makes parameter calibration more difficult. However, the metrics in Figure 11 show that Algorithm 3 converges faster than Algorithm 4 and often achieves better results.

One conspicuous feature is the high fluctuation in the calculation time per iteration in the applications of $\mathcal{N}_{\text{LSTM}}(\ \cdot\ ;\theta)$ during the optimisation when compared to Algorithm 1. On the one hand, this is due to the comparatively high complexity of the TLSTM. In this context high complexity means a high dimension $d \in \mathbb{N}$ of the parameter vector $\theta \in \mathbb{R}^d$. On the other hand, the main driver of the higher calculation time is the transformation given by equation (3.19) since on an algorithmic level the entire input sequence $\varphi_n, \varphi_{n+1}, \varphi_{n+2}, \varphi_{n+3}, \varphi_{n+4}$ has to be stored and transformed. In general, it becomes apparent at this point that the transformations in equations (3.9) and (3.19) are the critical aspects that compromise the performance of Algorithms 1 and 3.

Table 4 compares the performance of the three presented algorithms. In overall terms, Algorithm 3 achieves better compliance, whereas Algorithm 1 stands out due to its shorter calculation time.

method	applied load g	computation time	converges after n_{final}	evaluation of $J^\varepsilon(\varphi_{n_{\text{final}}})$	compliance $\int_{\Gamma_g} g \cdot u(\varphi)\,\mathrm{d}s$
Algorithm 4	$(0, -5000)^T$	4 min 43 sec	538	1475.39	1173.22
Algorithm 1	$(0, -5000)^T$	**4 min 05 sec**	441	**1206.49**	**1148.66**
Algorithm 3	$(0, -5000)^T$	4 min 34 sec	**425**	1209.24	1151.27
Algorithm 4	$(2632.16, -4251.09)^T$	4 min 31 sec	473	1487.39	1416.46
Algorithm 1	$(2632.16, -4251.09)^T$	**4 min 14 sec**	517	1475.05	1405.39
Algorithm 3	$(2632.16, -4251.09)^T$	4min 34 sec	**436**	**1209.24**	**1151.27**
Algorithm 4	$(-733.23, -4945.95)^T$	4 min 50 sec	**457**	1115.66	1062.41
Algorithm 1	$(-733.23, -4945.95)^T$	**4 min 21 sec**	497	**1049.91**	1042.58
Algorithm 3	$(-733.23, -4945.95)^T$	4 min 41 sec	484	1082.62	**1031.62**
Algorithm 4	$(-1099.92, -4877.25)^T$	5 min 11 sec	470	1042.18	992.28
Algorithm 1	$(-1099.92, -4877.25)^T$	**4 min 42 sec**	**447**	1048.44	998.34
Algorithm 3	$(-1099.92, -4877.25)^T$	4 min 43 sec	542	**1041.28**	**992.16**
Algorithm 4	$(-3197.16, -3844.24)^T$	2 min 17 sec	365	**845.93**	**805.94**
Algorithm 1	$(-3197.16, -3844.24)^T$	**2 min 13 sec**	**346**	852.36	811.15
Algorithm 3	$(-3197.16, -3844.24)^T$	2 min 59 sec	366	848.52	808.40

Table 4: Comparison of metrics between Algorithm 4, 1 and 3.

5 Discussion & Conclusions

The objective of this work is to devise neural network architectures that can be used for efficient topology optimisation problems. These tasks are computationally involved and typically are inevitably carried out with a large number of optimisation steps, each requiring (depending on the chosen method) the solution of state and adjoint equations to determine the gradient direction. Instead of learning a surrogate for state and adjoint equations, we present NN architectures that directly predict this gradient, leading to very efficient optimisation schemes. A noteworthy aspect of our investigation is to take into account uncertainties of the model data in a risk-averse optimisation formulation. This is a generalisation of the notion of “loading scenarios” that are commonly used in practice for a fixed set of parameter realisations. With our continuous presentation of uncertainties in the material and of the load acting on the considered structure, the robustness of the computed design with respect to these uncertainties can be controlled by the parameter of the CVaR used in the cost functional. Since computations with uncertainties require a substantial computational effort, our central goal is to extend the algorithms used in [9, 10] by introducing appropriate NN predictions, reducing the needed iteration steps. In contrast to other machine learning approaches, our aim is to achieve this even for adaptively adjusted finite element meshes since this has proven to be crucial for good performance in previous work. For this to work, an underlying sufficiently fine reference mesh is assumed for the training data and the prediction. Moreover, in contrast to other NN approaches for this problem, we consider the evolution of a continuous (functional) representation of a phase field determining the material distribution.

Ideally the NN architectures should speed up the deterministic topology optimisation problem and consequently also the risk-averse optimisation under uncertainties. This is achieved in Section 3.1 by embedding a CNN in the optimisation for both the deterministic and the stochastic setting.

The observed numerical results for a common 2D bridge benchmark are on par with the reference method presented in [10]. However, the gradient step predicted by the NN architectures allows for significantly larger iteration steps, rendering the optimisation procedure more efficient. This directly transfers to the Monte Carlo based risk-averse optimisation under uncertainties as defined by Algorithm 2 since the samples for the statistical estimation are obtained very cheaply. In addition to the CNN, a second architecture is illustrated in terms of a LSTM. This in general leads to a better quality of the optimisation and is motivated by the idea that a memory of previous gradients may lead to a more accurate prediction of the next gradient step. However, it comes at the cost of longer computation times due to the transformation between the different adapted computation meshes (see Section 4.4). Hence, a substantial performance improvement could be achieved by reducing the complexity of the transformations of equation (3.9) and (3.19).

There are several interesting research directions from the presented approach and observed numerical results. Regarding the chosen architectures, an interesting extension would be to consider graph neural networks (GNN) since there, the underlying mesh structure is mapped directly to the NN. Consequently, the costly transfer operators from current mesh to reference mesh of the design space could be alleviated, removing maybe the largest computational burden of our approach. Moreover, transformer architectures have probably superseded LSTMs and it would be worth examining this modern architecture in the context of this work.

The loss function used in the training also leaves room for improvements. For example, instead of the simple mean squared error used here, one could approximate the objective functional of equation (2.5) directly in the loss function. Regarding the training process, there are modern techniques to improve the efficiency and alleviate over-fitting such as early stopping, gradient clipping, adaptive learning rates and data augmentation as discussed in [18]. Moreover, transfer learning in a limited-data setting could substantially reduce the amount of training data required.

This work mainly serves as a proof of concept for treating the considered type of optimisation problems with modern NN architectures. An important step towards practicability is the further generalisation of this model, e.g., to arbitrary problems (with parameterised boundary data and constraints), determined by descriptive parameters drawn from arbitrary distributions according to the problem at hand. Moreover, the models presented here can be used as a basis for theoretical proofs (e.g., regarding the complexity of the representation) and further systematic experiments.

A Bridge benchmark problem

A.1 Deterministic bridge optimisation

For a comparison between Algorithm 4 from [9, 10] and its extension to NNs, we make use of a bridge benchmark problem at selected locations. The name comes from the optimal shape resembling a bridge, which exhibits the best stiffness under the given constraints and forces acting on it. To be specific, the parameters are given as follows: Assume design domain $D = [-1, 1] \times [0, 1]$ with boundaries $\Gamma_D = [-1, -0.9] \times \{0\}, \Gamma_g = [-0.02, 0.02] \times \{0\}$ on which the load $g = (0, -5000)^T$ is applied and the slip condition $\Gamma_S = [0.9, 1.0] \times \{0\}$ is set. The Lamé coefficients are given by $\mu_{\text{mat}} = \lambda_{\text{mat}} = 150$. Furthermore, the volume constraint is $m = 0.4$ and $\varepsilon = \frac{1}{16}$. This is the same setting as in the deterministic experiment in [9, 10].

The initial material distribution is given by $\varphi_0(x) = 0.5 \ \forall x \in D$. Several iteration results of φ_n from Algorithm 4 are depicted in Figure 12. As can be seen by the finer edges in the images, in the course of optimising the topology (compare, e.g., φ_{200} and φ_{538}), an adapted mesh is used which is refined depending on φ_n in order resolve fine details of the topology and to save computational costs (see Section B).

Algorithm 4 took 538 iterations to converge. Figure 13 illustrates other metrics in the optimisation. The convergence of $J^\varepsilon(\varphi)$ is clearly visible. Through the adaptation method of the step size τ_n (see [9]), it becomes increasingly larger when φ_n begins to converge towards the optimal mesh $\mathcal{T}_m$. The small spikes in all time series are due to the refinement of the mesh $\mathcal{T}_m$. In the lower-right corner, it can be seen that the calculation time increases with the fineness of the mesh $\mathcal{T}_m$. The lower-left part of Figure 13 shows γ_n, which stabilises the form of φ_n, but plays no further role in our investigations.

A.2 Stochastic bridge problem

This modification of the experiment described in Section A.1 introduces uncertainties in the data, which render the problem much more involved. The LaméâĂŞcoefficients $\mu_{\text{mat}} = \lambda_{\text{mat}}$ are modelled as a truncated lognormal KarhunenâĂŞLoÃĺve expansion with 10 modes, a mean value of 150 and a covariance length of 0.1 which is scaled by a factor of 100. The load g is assumed as a vector with mean $(0, -5000)$ and a random rotation angle simulated through a truncated normal distribution with bounds $[\frac{-\pi}{2}, \frac{\pi}{2}]$, standard deviation 0.3 and mean 0. In each iteration we use $N = 224$ samples for the evaluation of the risk functional.

Some of the resulting iterations of the optimisation process are depicted in Figure 14. By calculating the expected value of the functional in equation (2.7) (with parameter $\beta = 0$), one can see a loss of symmetry in the resulting topology compared to the deterministic setting from Example 3.2 since the load is almost always not perpendicular to the load bearing boundaries. The main difference is the strain on the Dirichlet boundary, which is introduced by the moved left most spoke. In contrast to this, the right-hand side closely resembles the deterministic case since the slip boundary cannot absorb energy in the tangential direction. In this particular stochastic setting, an additional spoke is formed.

Algorithm 4: Deterministic optimisation algorithm from [9].

```
Input: mesh $\mathcal{T}_0$ and initial values $\varphi_0$
for $m = 0, 1, \dots$ until converged do
    for $n = 0, 1, \dots$ until converged do
        solve state equation on mesh $\mathcal{T}_m \Rightarrow u_{n+1}$
        solve adjoint equation on mesh $\mathcal{T}_m \Rightarrow p_{n+1}$
        solve gradient equation on mesh $\mathcal{T}_m \Rightarrow \varphi^*_{n+1}$
        project $\varphi^*_{n+1}$ to $[0, 1] \Rightarrow \varphi_{n+1}$
    end
    adapt mesh according to Section B $\Rightarrow \mathcal{T}_{m+1}$
end
```

Algorithm 5: Stochastic optimisation algorithm from [9].

```
Input: mesh $\mathcal{T}_0$ and initial values $\varphi_0$
for $m = 0, 1, \dots$ until converged do
    for $n = 0, 1, \dots$ until converged do
        for $i = 1, \dots, N$ do
            sample $g(\omega_i), \lambda_{mat}(\omega_i), \mu_{mat}(\omega_i), \omega_i \in \Omega$
            solve state equation on mesh $\mathcal{T}_m \Rightarrow u_{n+1}(\omega_i)$
            solve adjoint equation on mesh $\mathcal{T}_m \Rightarrow p_{n+1}(\omega_i)$
            solve gradient equation on mesh $\mathcal{T}_m \Rightarrow \varphi^*_{n+1}(\omega_i)$
        end
        compute the mean $\hat{\varphi}_{n+1} = \frac{1}{N}\sum_{i=1}^{N} \varphi^*_{n+1}(\omega_i)$
        project $\hat{\varphi}_{n+1}$ to $[0, 1] \Rightarrow \varphi_{n+1}$
        adapt mesh according to Section B $\Rightarrow \mathcal{T}_{m+1}$
    end
end
```

B Finite element discretization

The physical space $D \subset \mathbb{R}^2$ is discretised with first order conforming finite elements with the `FEniCS` framework [2]. The reason for the favourable performance of the optimiser from [9, 10] is the adaptive mesh refinement based on gradient information of the phase field. The idea is that the optimisation is started on a coarse mesh and refined in the course of the optimisation depending on the topology (more precisely on the phase transitions of φ). For this purpose it is assumed that the domain D is a convex polygon and is described with first order conforming elements by the mesh $\mathcal{T}_m = (V(\mathcal{T}_m), E(\mathcal{T}_m))$ at iteration step $m \in \mathbb{N}$, which also can be understood as a graph. The mesh consists of triangles $T \in \mathcal{T}_m$ and an associated set of edges $E(T) \subset E(\mathcal{T}_m) \subset D \times D$

and vertices $V(T) \subset V(\mathcal{T}_m) \subset D$. Based on this mesh, the next mesh $\mathcal{T}_{m+1}$ is generated with a simple error indicator using the bulk criterion for the DÃűrfler marking [8] to determine which triangles T should be refined. Since φ moves within the domain D, the mesh refinement necessarily reflects this. So instead of just refining the previous mesh, for every refinement we start with the initial mesh $\mathcal{T}_0$, interpolate the current solution and displacement onto that mesh, refine according to the associated indicators and interpolate the current solutions φ_n and u_n onto the finer mesh. We repeat this process until a mesh $\mathcal{T}_{m+1}$ is obtained that is adequately finer than $\mathcal{T}_m$. This refinement takes place whenever φ_n converges on the current mesh $\mathcal{T}_m$. The refinement across an optimisation with Algorithm 4 is shown in Figure 15. It can be observed that the mesh is refined along the edges of φ. This dynamic characterisation (depending on φ and thus on the coefficients of the associated PDE) of the domain by the mesh $\mathcal{T}_m$ poses a special challenge for the presented NN architectures when solving equation (2.6) or (2.11), respectively. The relative change $e_n := \frac{\|\varphi_{n+1}-\varphi_n\|_{L^2(D)}}{\|\varphi_{n+1}\|_{L^2(D)}}$ in combination with the step size τ_n is used as convergence criterion, i.e.,

$$\frac{e_n}{\tau_n} < \varepsilon, \qquad \varepsilon > 0. \tag{B.1}$$

This means that as soon as φ on a mesh $\mathcal{T}_m$ does not change significantly in relation to the step size τ_n, ϕ_n is considered converged. The refinement of the mesh is bounded by a chosen maximum of vertices $|V(\mathcal{T}_m)| \leq b \in \mathbb{N}$.

We understand $\varphi_n \in \mathbb{R}^{|(\mathcal{T}_m)|}$ and $u_n \in \mathbb{R}^{d\times|(\mathcal{T}_m)|}$ as discretisation on $\mathcal{T}_m$ at iteration step $n \in \mathbb{N}$ in the sense that the value at every node $v_i \in V(\mathcal{T}_m),\ i \leq |V(\mathcal{T}_m)|$ is evaluated according to

$$u_n^i := u_n(v_i) \quad \text{or} \quad \varphi_n^i := \varphi_n(v_i)$$

in this node.

C Additional experiments

Figures 16 and 17 illustrate numerically less robust TCNN predictions for large gradient step sizes as mentioned in Section 4.1.2. In Figure 18, some iterations of a classical optimisation in comparison with the CNN assisted optimisation are depicted, showing basically identical results. However, it also can be observed that the distribution of the material converges faster when using the CNN. After 100 iterations, the first spokes for stabilising the arc can already be seen.

References

[1] Diab W Abueidda, Seid Koric, and Nahil A Sobh. Topology optimization of 2D structures with nonlinearities using deep learning. *Computers & Structures*, 237:106283, 2020.

[2] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie Rognes, and Garth Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[3] Gorkem Can Ates and Recep M Gorguluarslan. Two-stage convolutional encoder-decoder network to improve the performance and reliability of deep learning models for topology optimization. *Structural and Multidisciplinary Optimization*, 63(4):1927–1950, 2021.

[4] Ruijin Cang, Hope Yao, and Yi Ren. One-shot optimal topology generation through theory-driven machine learning. *CoRR*, abs/1807.10787, 2018.

[5] Aaditya Chandrasekhar and Krishnan Suresh. Tounn: topology optimization using neural networks. *Structural and Multidisciplinary Optimization*, 63(3):1135–1149, 2021.

[6] Hao Deng and Albert C To. A parametric level set method for topology optimization based on deep neural network. *Journal of Mechanical Design*, 143(9), 2021.

[7] Tim Dockhorn. A discussion on solving partial differential equations using neural networks. *CoRR*, abs/1904.07200, 2019.

[8] Willy Dörfler. A convergent adaptive algorithm for poisson's equation. *SIAM Journal on Numerical Analysis*, 33(3):1106–1124, 1996.

[9] Martin Eigel, Johannes Neumann, Reinhold Schneider, and Sebastian Wolf. Stochastic topology optimisation with hierarchical tensor reconstruction. 2362, 2016.

[10] Martin Eigel, Johannes Neumann, Reinhold Schneider, and Sebastian Wolf. Risk averse stochastic structural topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 334:470–482, 2018.

[11] Ebrahim Ghaderpour, Spiros D Pagiatakis, and Quazi K Hassan. A survey on change detection and time series analysis with applications. *Applied Sciences*, 11(13):6141, 2021.

[12] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[13] Alex Halle, Lucio Flavio Campanile, and Alexander Hasse. An artificial intelligence–assisted design method for topology optimization without pre-optimized training data. *Applied Sciences*, 11(19):9041, 2021.

[14] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[16] Nikos Ath. Kallioras and Nikos D. Lagaros. Dl-scale: Deep learning for model upgrading in topology optimization. *Procedia Manufacturing*, 44:433–440, 2020. The 1st International Conference on Optimization-Driven Architectural Design (OPTARCH 2019).

[17] Manoj Malviya. A systematic study of deep generative models for rapid topology optimization. 2020.

[18] Raoof Naushad, Tarunpreet Kaur, and Ebrahim Ghaderpour. Deep transfer learning for land use and land cover classification: A comparative study. *Sensors*, 21(23):8083, 2021.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[20] Sharad Rawat and M.-H. Herman Shen. A novel topology optimization approach using conditional deep learning. *CoRR*, abs/1901.04859, 2019.

[21] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

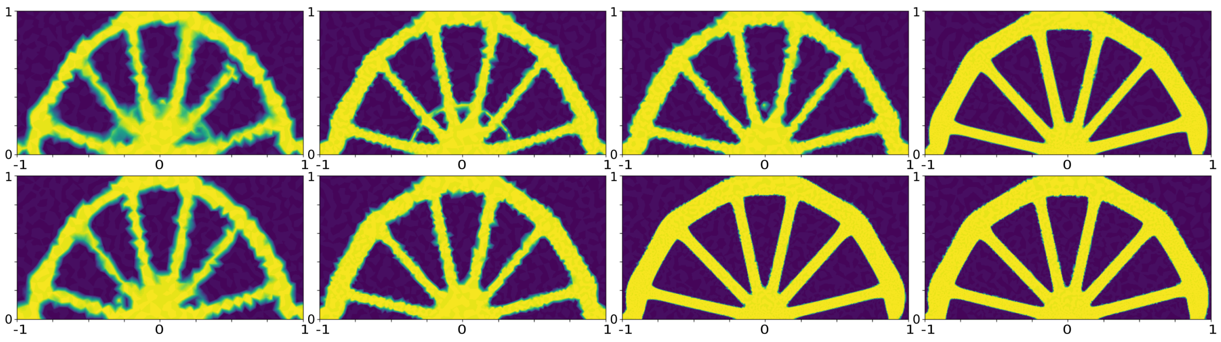
[22] WS Slaughter and J Petrolito. Linearized Theory of Elasticity. *Applied Mechanics Reviews*, 55(5):B90–B91, 09 2002.

[23] Ivan Sosnovik and Ivan Oseledets. Neural networks for topology optimization. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 34(4):215–223, 2019.

[24] Dalei Wang, Cheng Xiang, Yue Pan, Airong Chen, Xiaoyi Zhou, and Yiquan Zhang. A deep convolutional neural network for topology optimization with perceptible generalization ability. *Engineering Optimization*, 54(6):973–988, 2022.

[25] Daniel A White, William J Arrighi, Jun Kudo, and Seth E Watts. Multiscale topology optimization using neural network surrogate models. *Computer Methods in Applied Mechanics and Engineering*, 346:1118–1135, 2019.

[26] Yiquan Zhang, Airong Chen, Bo Peng, Xiaoyi Zhou, and Dalei Wang. A deep convolutional neural network for topology optimization with strong generalization ability. *CoRR*, abs/1901.07761, 2019.



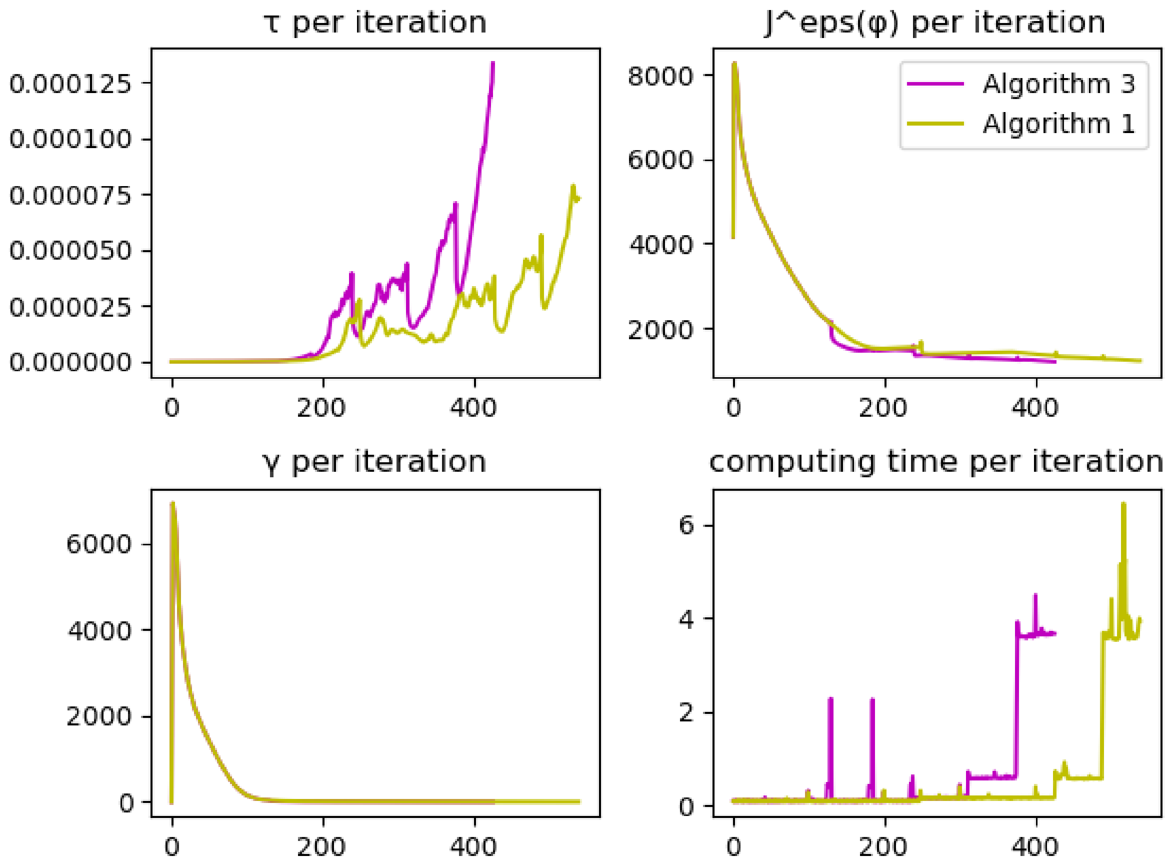


Figure 11: Comparison of $\varphi_{200}, \varphi_{300}, \varphi_{400}, \varphi_{n_{\text{final}}}$ between Algorithm 4 (top row) and 3 (bottom row).

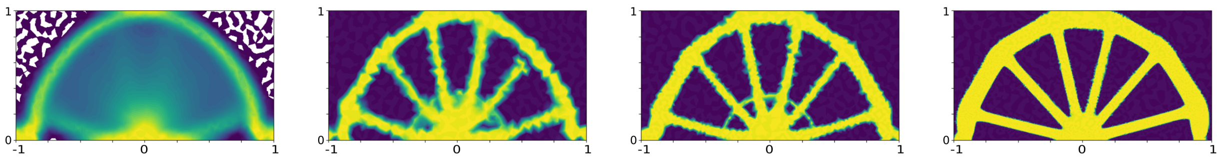


Figure 12: Iterations $\varphi_{100}, \varphi_{200}, \varphi_{300}, \varphi_{538}$ from Algorithm 4.

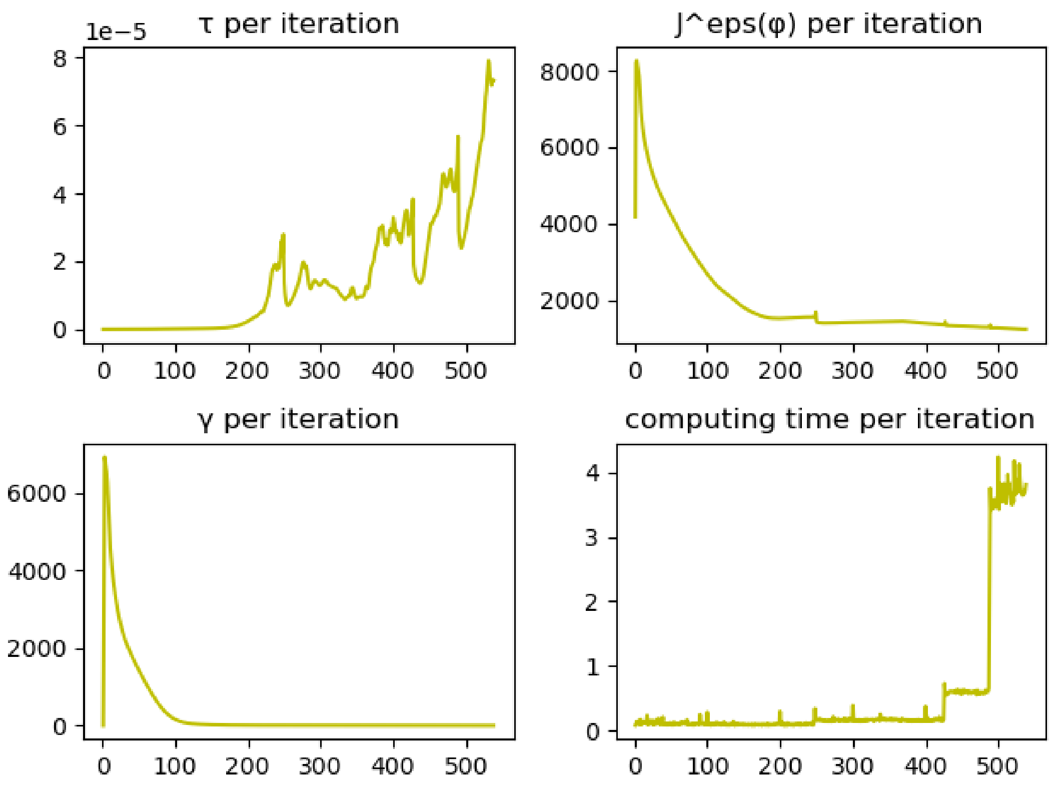


Figure 13: Metrics of the optimisation of equation (2.4) using Algorithm 4.

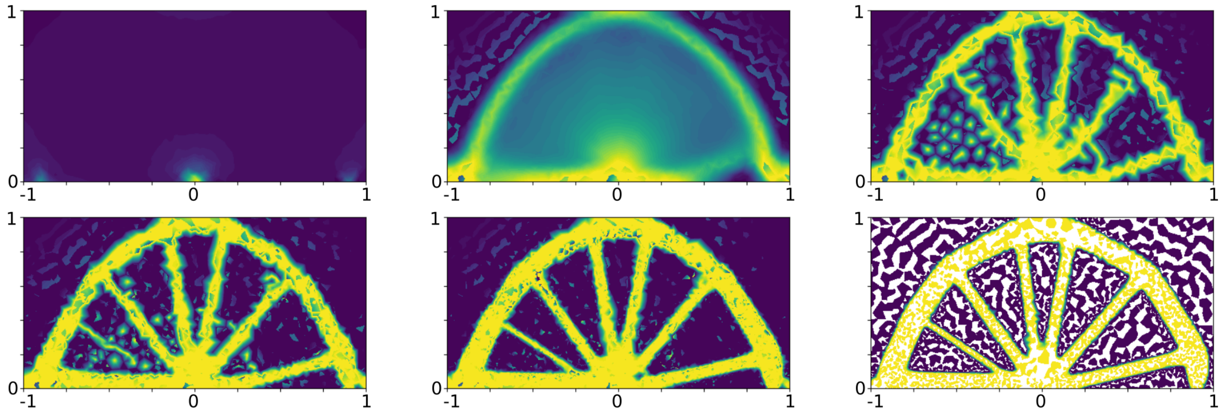


Figure 14: Iterations $\varphi_1, \varphi_{100}, \varphi_{200}, \varphi_{300}, \varphi_{400}, \varphi_{490}$ from Algorithm 5.

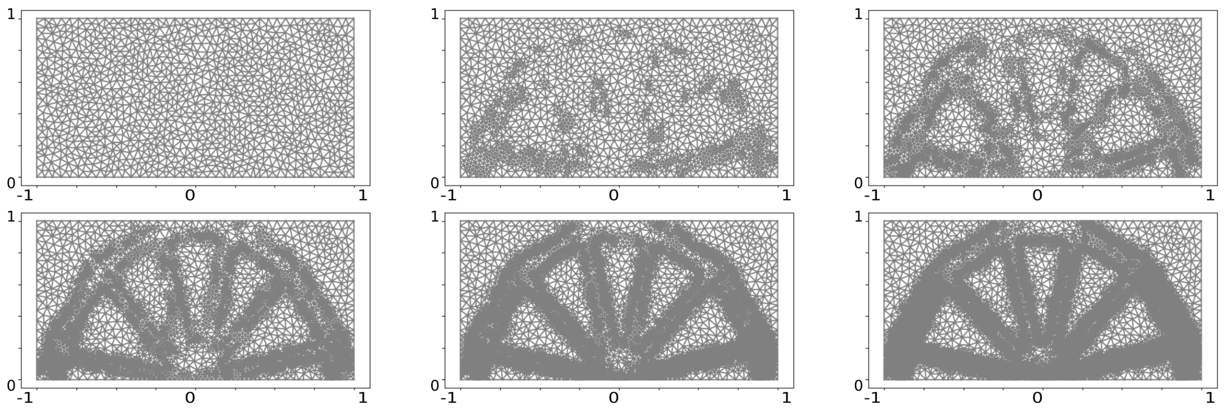


Figure 15: Iteration of adaptive mesh $\mathcal{T}_m$ from Section A.1.

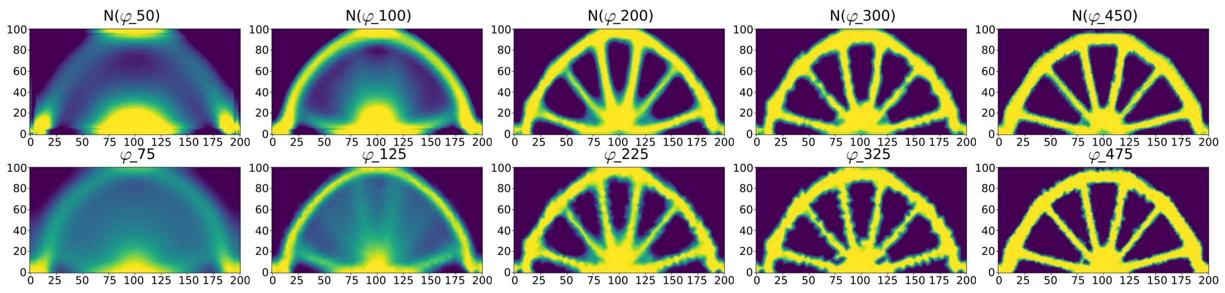


Figure 16: $\mathcal{N}^{25}_{\text{CNN}}(\varphi_n, u_{n+1}; \theta)$ (top row) in comparison with φ_{n+25} (bottom row).

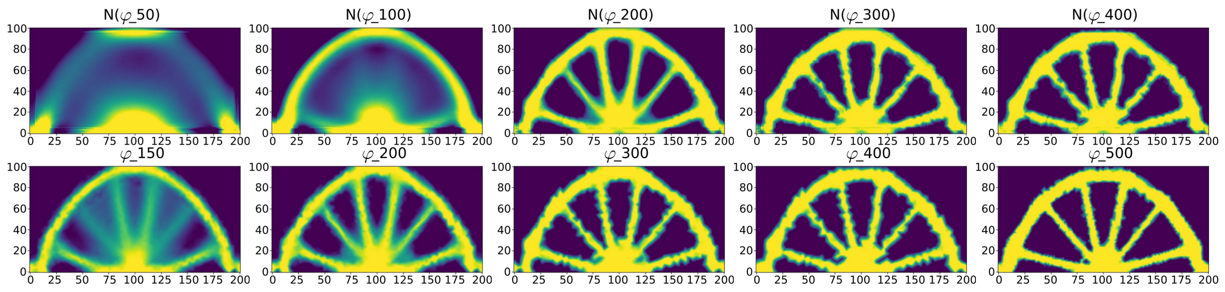
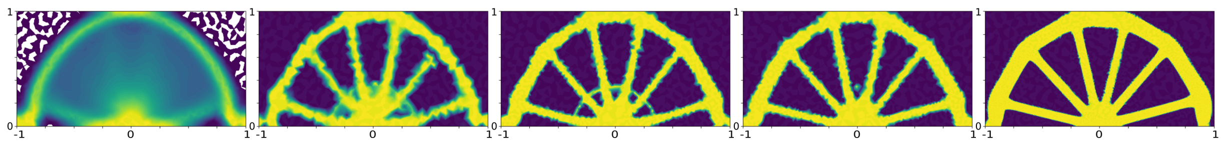
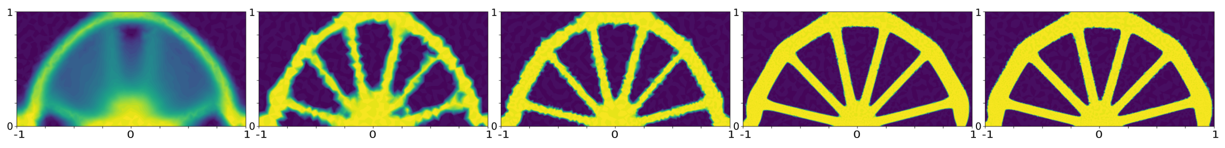


Figure 17: $\mathcal{N}^{100}_{\text{CNN}}(\varphi_n, u_{n+1}; \theta)$ (top row) in comparison with φ_{n+100} (bottom row).



(a) Sequence $\varphi_{100}, \varphi_{200}, \varphi_{300}, \varphi_{400}, \varphi_{538}$ from Algorithm 4



(b) Sequence $\varphi_{100}, \varphi_{200}, \varphi_{300}, \varphi_{400}, \varphi_{441}$ from Algorithm 1.

Figure 18: Optimisation without and with $\mathcal{N}_{\text{CNN}}$.