IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 22 July 2024, accepted 30 August 2024, date of publication 23 September 2024, date of current version 1 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3465572

RESEARCH ARTICLE

Sobolev Neural Network With Residual Weighting as a Surrogate in Linear and Non-Linear Mechanics

A. O. M. KILICSOY[1], J. LIEDMANN[2], M. A. VALDEBENITO[1], F.-J. BARTHOLD[2], AND M. G. R. FAES[1]

[1]Chair for Reliability Engineering, Fakultät Maschinenbau, Technical University Dortmund, 44227 Dortmund, Germany
[2]Lehrstuhl Baumechanik, Fakultät Architektur und Bauingenieurwesen, Technical University Dortmund, 44227 Dortmund, Germany

Corresponding author: A. O. M. Kilicsoy (ali.kilicsoy@tu-dortmund.de)

This work was supported by German Research Foundation (DFG) on "Damage Controlled Forming Processes" under Grant TRR 188-278868966.

ABSTRACT Areas of computational mechanics such as uncertainty quantification and optimization usually involve repeated evaluation of numerical models that represent the behavior of engineering systems. In the case of complex non-linear systems however, these models tend to be expensive to evaluate, making surrogate models quite valuable. Artificial neural networks approximate systems very well by taking advantage of the inherent information of its given training data. In this context, this paper investigates the improvement of the training process by including sensitivity information, which are partial derivatives w.r.t. inputs, as outlined by Sobolev training. In computational mechanics, sensitivities can be applied to neural networks by expanding the training loss function with additional loss terms, thereby improving training convergence resulting in lower generalisation error. This improvement is shown in two examples of linear and non-linear material behavior. More specifically, the Sobolev designed loss function is expanded with residual weights adjusting the effect of each loss on the training step. Residual weighting is the given scaling to the different training data, which in this case are response and sensitivities. These residual weights are optimized by an adaptive scheme, whereby varying objective functions are explored, with some showing improvements in accuracy and precision of the general training convergence.

INDEX TERMS Machine learning, Sobolev training, residual weighting, finite element modelling, linear and non-linear mechanics, neural networks, optimization, surrogate model.

I. INTRODUCTION

In multiple disciplines of engineering, various system analysis methods exist to predict their functional behavior. Especially numerical techniques, such as finite element method (FEM) models, are a fundamental tool for various engineering tasks. In particular, when quantifying uncertainty and performing optimization, repeated evaluations of these models are necessary [1]. Such repeated evaluations, however, scale poorly with system size and complexity. Artificial neural networks (ANN) present an opportunity to replace time-intensive methods as a surrogate model [2], [3] and have been successfully applied to the field of uncertainty quantification [4], [5], [6]. While the first version of ANNs, the Perceptron, dates far back [7], a surge of research has allowed ANNs to perform incredibly well on tasks previously too complex to solve by clear and definite methods with satisfactory results [8], [9], [10], [11]. In its most basic form, ANNs are capable of classification and regression, which in the case of the latter makes them general approximators [12]. There has also been research on intertwined processing between first principle models and neural networks [13], [14]. In the recent paper [15], the intrusive method of using the material and force tensor of a system used during FEM approximations, and applying these in the loss of a neural network is shown to evaluate

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

the system better. In any case, training a neural network can be a challenging process which has been approached in regards to the training algorithm [16], [17], [18], training data [19], [20] and loss function design [21], [22]. In this work, specifically the adjustment of the loss function in order to introduce sensitivities, is of interest, as sensitivities bring additional information with them, which can be used to improve the training of models [23], [24]. The use of gradient data to improve the performance of neural networks has been applied in a handful of papers and in the case of available sensitivity data, gradient-enhancement is an essential and simple step. Usually, these sensitivities can be computed efficiently through adjoint methods, but these can be evaluated directly as well [25]. The paper coining the term Sobolev training [26], shows the use of gradient data linked to Sobolev spaces, naming the models as Sobolev artificial neural network, or SANN. The paper goes to prove the usability of gradient information through Sobolev space for a small, simple neural network. Similar in essence, application of Lipschitz continuity during training can lead to an increase in robustness of the model, coined as Sobolev regularization [27], [28]. In another paper [29], the use of gradient data of a partial differential equation (PDE) when considering a physics informed neural network [30], named gradient-enhanced PINN, or gPINN, improves the accuracy and training performance of a PINN, applying it on a few mathematical function examples. In any case, Sobolev training is shown to produce better accuracy and training convergence than basic training, where only the model response is considered. In regards to correctly weighting the Sobolev training losses, another paper expands on SANN by introducing weighting of the individual losses introduced by Sobolev training, by applying a fixed linear increase of the weight attributed to sensitivities, thereby improving performance, consequently naming it mSANN, or modified Sobolev artificial neural network [31]. These expansions of ANNs allow for greater accuracy and faster training, taking ANNs a step further as surrogate models replacing established numerical methods [32]. In this work, the established Sobolev training is applied to approximate a mechanical problem of linear and non-linear nature, whereby a finite element model employing variational sensitivity analysis is used to compute responses and sensitivities needed for training of the ANN [33], [34]. More specifically, this gradient-enhanced neural network focuses on surrogate modelling the finite element model which uses variational sensitivity analysis of a mechanical system of linear and non-linear elasticity material behavior [35]. Because various model designs are generally task specific, application to real physical tasks is important for further insights. Additionally, application of Sobolev training with larger neural networks compared to e.g. the Sobolev training paper, but still small compared to deep neural networks, will provide new insights. In contrast to PINNs, we only base the training of the Sobolev neural network on responses and their sensitivities that are obtained by performing variational finite element simulations, which is non-intrusive to the general neural network architecture. For this, the loss function of the neural network is expanded with additional losses for each sensitivity, by which the model parameters are optimized in order for the backpropagated gradients to approximate the given sensitivities. Lastly, in this work, this newly designed loss function consisting of multiple losses, will be improved by residual weights. The residual weights will scale the individual loss terms which correspond to the response and sensitivities of the training data. The goal of this residual weighting is the optimization of the Sobolev training convergence leading to faster training or lower approximation error. This weighting task has been explored within mSANN, but also in other papers, where different algorithms and methods of optimization are applied [36], [37]. In the same non-intrusive way as mSANN, the loss function will be expanded through residual weights corresponding to each individual loss. However in this work, an adaptive scheme to optimize the residual weights is applied, which could potentially improve Sobolev training further, in contrast to general parameter tuning such as in mSANN. This adaptive scheme optimizes the residual weights in a parallel process during training of the neural network, for their own defined objective function. A set of objective functions are chosen based on prior beliefs of the system setup by the multiple individual losses. The different methods are tested and then compared and analysed. With this framework set, the paper will be structured as entailed in the following. Section II will give a brief theoretical overview of the fundamentals this work is based on, such as the FEM and ANNs. Section III of this work gives an overview of Sobolev training as it has already been done and then links it to the developments of residual weighting in detail. Section IV describes the mechanical use case simulated by the FEM and the details on the programming of the neural networks. Section V presents the results of the residual weighting methods for further analysis. The following section VI covers the analysis and discussion of the results. Section VII concludes the work and its results, giving a short outlook on potential areas of further study.

II. THEORY

A. GENERAL REMARKS

This section will setup a foundation of the basic applied FEM and neural network. This is done by elaborating the core formulation behind the variational finite element simulation providing the training data and the basic neural network calculations involved during training and prediction. In regards to the neural network core terms are explained.

B. ANALYSIS MODEL

We make use of a finite element model which models a physical problem of linear or non-linear material behavior. The finite element method is a numerical tool for solving underlying partial differential equations of engineering tasks. In this work, the solution to the weak equilibrium equation

for non-linear elasticity in the reference configuration $\mathcal{B}_0$ is demanded

$$R(\boldsymbol{u},\boldsymbol{v}) = \int_{\mathcal{B}_0} \boldsymbol{P}^K(\boldsymbol{u}) : \nabla \boldsymbol{v}\, dV - \int_{\mathcal{B}_0} \boldsymbol{b}_0 \cdot \boldsymbol{v}\, dV - \int_{\mathcal{B}_0} \boldsymbol{t}_0 \cdot \boldsymbol{v}\, dA = 0, \tag{1}$$

where $\boldsymbol{P}^K$ is the first Piola-Kirchhoff stress tensor, $\boldsymbol{b}_0$ and $\boldsymbol{t}_0$ denote body and traction forces, respectively. Further, $\boldsymbol{u}$ denotes the vector of primary displacements and $\boldsymbol{v}$ is the vector of test functions. Due to the non-linear system response, a Newton-Raphson procedure is employed to iteratively solve (1), which implies its linearization

$$R(\boldsymbol{u},\boldsymbol{v}) + \delta_u R(\boldsymbol{u},\boldsymbol{v};\,\Delta\boldsymbol{u}) = 0. \tag{2}$$

In the FEM a system is approximated through various techniques. In a process called discretization, the observed system geometry is subdivided into multiple simple sections called finite elements, which make up the complete system mesh. This allows to numerically solve the weak form of equilibrium. The finite element system is set up through the use of known shape functions - traditionally chosen as polynomials - for the approximation of geometry, displacements and test functions. Following the isoparametric concept, the same shape functions are used for all approximations in each finite element. With the discrete approximations

$$R(\boldsymbol{u},\boldsymbol{v}) \approx \boldsymbol{v}^T\boldsymbol{R} \text{ and } \delta_u R(\boldsymbol{u},\boldsymbol{v},\Delta\boldsymbol{u}) \approx \boldsymbol{v}^T\boldsymbol{K}\Delta\boldsymbol{u}, \tag{3}$$

where $\boldsymbol{R}$ and $\boldsymbol{K}$ denote the discrete residual vector and tangent stiffness matrix, respectively, and excluding the trivial solution $\boldsymbol{v} = \boldsymbol{0}$, the discrete version of (2) solved for the unknown displacement increment vector $\Delta\boldsymbol{u}$ in each Newton-Raphson iteration reads

$$\boldsymbol{K}\Delta\boldsymbol{u} = -\boldsymbol{R}. \tag{4}$$

Further details on finite element analysis (FEA) can be found e.g. in [38], [39], [40], [41], and [42] to name a few. In design sensitivity analysis, generally, the change of a response function $f(\boldsymbol{u}(\boldsymbol{x}),\boldsymbol{x})$ w.r.t. a change in chosen design or model parameters $\boldsymbol{x}$ defining the model design shall be quantified. Utilising variational sensitivity analysis, cf. e.g. [33] and [34], this change can be written in variational form

$$\delta f = \delta_u f + \delta_x f = \left[\frac{\partial f}{\partial \boldsymbol{u}}\right]\delta\boldsymbol{u} + \left[\frac{\partial f}{\partial \boldsymbol{x}}\right]\delta\boldsymbol{x}. \tag{5}$$

Further, considering that a design change $\delta\boldsymbol{x}$ must not violate the equilibrium of the system, i.e.

$$\delta R(\boldsymbol{u},\boldsymbol{v},\delta\boldsymbol{u},\delta\boldsymbol{x}) = \delta_u R(\boldsymbol{u},\boldsymbol{v},\delta\boldsymbol{u}) + \delta_x R(\boldsymbol{u},\boldsymbol{v},\delta\boldsymbol{x}) = 0,$$

and approximate both variations using

$$\delta_u R(\boldsymbol{u},\boldsymbol{v},\delta\boldsymbol{u}) \approx \boldsymbol{v}^T\boldsymbol{K}\delta\boldsymbol{u} \text{ and } \delta_x R(\boldsymbol{u},\boldsymbol{v},\delta\boldsymbol{x}) \approx \boldsymbol{v}^T\boldsymbol{P}\delta\boldsymbol{x}, \tag{6}$$

where $\boldsymbol{P}$ represents the tangent pseudoload matrix, the total response sensitivity can be identified by rearranging the discrete total variation of the weak equilibrium condition, viz.

$$\boldsymbol{K}\delta\boldsymbol{u} = -\boldsymbol{P}\delta\boldsymbol{x}. \tag{7}$$

This procedure is commonly referred to as the direct differentiation method and results in the computation of the sensitivity matrix $\boldsymbol{S}$ by solving (7) for the unknown displacement variations, i.e

$$\delta\boldsymbol{u} = -\boldsymbol{K}^{-1}\boldsymbol{P}\delta\boldsymbol{x} = \boldsymbol{S}\delta\boldsymbol{x}. \tag{8}$$

The response function of interest in this study is the so called stress triaxiality defined as the ratio of the mean stress σ_{Mean} and equivalent von Mises stress σ_{Mises}

$$T = \frac{\sigma_{\text{Mean}}}{\sigma_{\text{Mises}}} = \frac{\operatorname{tr}\boldsymbol{\sigma}}{3\sqrt{3J_2}}, \tag{9}$$

where J_2 is the second deviatoric stress invariant and $\operatorname{tr}\boldsymbol{\sigma}$ is the stress tensor. Therefore, it characterizes the current stress state as follows. A high positive stress triaxiality corresponds to a tensile stress state, while negative values indicate a compression stress state. A low magnitude corresponds to a shear dominated stress state.

Utilizing the above described variational method, the discrete sensitivity relation of the stress triaxiality can be derived to

$$\delta T = \left[\frac{\partial T}{\partial \sigma_{\text{Mean}}}\frac{\partial \sigma_{\text{Mean}}}{\partial \boldsymbol{\sigma}} + \frac{\partial T}{\partial \sigma_{\text{Mises}}}\frac{\partial \sigma_{\text{Mises}}}{\partial \boldsymbol{\sigma}}\right]\delta\boldsymbol{\sigma}, \tag{10}$$

where the determination of the partial derivatives are straight forward and the total variation of the stress tensor can by means of (5) and (8) be identified to

$$\delta\boldsymbol{\sigma} = \left[\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{u}}\boldsymbol{S} + \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{x}}\right]\delta\boldsymbol{x}. \tag{11}$$

Further details on derivation and implementation can be found e.g. in the referenced literature. A non-linear compressible Neo-Hookean strain energy function of the form

$$W^{NH} = \frac{1}{2}\mu(I_c - 3 - 2\log(J)) + \frac{1}{2}\boldsymbol{\lambda}(J-1)^2 \tag{12}$$

is used for the computation of the stress response, where μ and $\boldsymbol{\lambda}$ are the elastic Lamé parameters, $I_c = \operatorname{tr}(\boldsymbol{C}) = \operatorname{tr}(\boldsymbol{F}^T\boldsymbol{F})$ is the first invariant of the left Cauchy-Green deformation tensor and $J = \det(\boldsymbol{F})$ denotes the determinant of the deformation gradient $\boldsymbol{F}$.

Remark: Note that within the work at hand the design parameters are chosen as geometric control point coordinates of a mesh controlling Bezier surface. However, within the FEM the only geometric information available is the nodal mesh coordinates. Therefore, a design velocity matrix is used to connect the nodal mesh coordinates with the underlying Bezier geometry description, cf. e.g. [43], [44], and [45].

C. ARTIFICIAL NEURAL NETWORKS

The neural network models in this paper have a basic structure and design of a regression feedforward model. According to the Universal Approximation Theorem [12] and its subsequent variations, neural networks are capable of approximating any continuous functions. ANNs have been proven to provide accurate results empirically. Following, the design parameters are referred to as the inputs $\boldsymbol{x}$ and the triaxiality $\boldsymbol{T}$ as true outputs $\boldsymbol{y}$.

1) PREPROCESSING

In supervised training, each input $\boldsymbol{x}_d$ is paired with a true response $\boldsymbol{y}_d$ for a given training data size d, whereby input and output usually contain multiple elements, denoted by subscript e. In general, the training data is split into multiple parts, which are called minibatches and each training iteration uses a single minibatch in order to minimize overfitting. The order of these minibatches is shuffled during training which adds a stochastic effect to the training process. When applying training data to a neural network, it is generally recommended to preprocess the data to avoid general pitfalls, such as exploding and vanishing gradients. In certain cases, trimming the data of outliers for numerical instability and normalization or standardization are important tools for training neural networks. Input and response data $\boldsymbol{x}_d, \boldsymbol{y}_d$ can be standardized by using the mean values $\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}$ and dividing by the corresponding standard deviations of each element $\boldsymbol{\sigma_x}, \boldsymbol{\sigma_x}$ respectively per Hadamard division:

$$\boldsymbol{x}_{d,s} = \frac{\boldsymbol{x}_d - \bar{\boldsymbol{x}}}{\boldsymbol{\sigma_x}} \tag{13}$$

$$\boldsymbol{y}_{d,s} = \frac{\boldsymbol{y}_d - \bar{\boldsymbol{y}}}{\boldsymbol{\sigma_y}} \tag{14}$$

Generally, it is necessary to preprocess the data with the to be approximated system in mind. The following sections will leave out subscript d, e, s for the sake of simplicity.

2) FORWARD PASS

For all equations relating to neural networks in this section, the number of neurons is defined as n. To provide a better overview in equations, the current layer from which anything is referenced is denoted as l, the following layer as $l+1$ and the previous layer is denoted as $l-1$. In addition, layers are always denoted as superscripts in brackets, that is $(\cdot)^{[\cdot]}$. The neurons are referenced with i, when referring to the current layer, and with j when referring to the previous layer. This numbering is denoted as subscripts, whereby in general j follows after i in the subscript. The linear term or the weighted sum, of a neuron, is called z, whereby w denotes a weight and b denotes a bias. The non-linear output of a node is called o, whereby the non-linear function applied in it, is notated as $a(\cdot)$. The input training data is referenced as x, the response training data or the true output as y and the neural network model output as $\hat{y}$. Every x is paired with a y, as supervised training is applied to the neural network models. However for the sake of overview, data size and minibatches will not be considered in the following equations and they apply to a single data pair of x, y. Still, every equation is applicable to multiple training data samples for a single iteration by following the defined equations in this section for each training data pair of the minibatch and simply taking the average of the final result, the loss $\mathcal{L}$. First, we define the weight matrices $\boldsymbol{W}^{[l]} \in R^{n^{[l]} \times n^{[l-1]}}$ for any layer l as:

$$\boldsymbol{W}^{[l]} = \begin{bmatrix} w_{1,1}^{[l]} & \cdots & w_{n^{[l-1]},1}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{1,n^{[l]}}^{[l]} & \cdots & w_{n^{[l-1]},n^{[l]}}^{[l]} \end{bmatrix} \tag{15}$$

The bias vector $\boldsymbol{b}^{[l]} \in R^{n^{[l]} \times 1}$ follows the following pattern for any layer l:

$$\boldsymbol{b}^{[l]} = \begin{pmatrix} b_{1,1}^{[l]} \\ \vdots \\ b_{1,n^{[l]}}^{[l]} \end{pmatrix} \tag{16}$$

The output vectors $\boldsymbol{o}^{[l]} \in R^{n^{[l]}}$ for any layer are defined as:

$$\boldsymbol{o}^{[l]} = \begin{pmatrix} o_1^{[l]} \\ \vdots \\ o_{n^{[l]}}^{[l]} \end{pmatrix} \tag{17}$$

The vectors $\boldsymbol{z}^{[l]} \in R^{n^{[l]}}$ containing the linear terms are defined as:

$$\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]} \cdot \boldsymbol{o}^{[l-1]} + \boldsymbol{b}^{[l]} \tag{18}$$

whereby the vectors $\boldsymbol{o}^{[l-1]}$ contain the non-linear outputs of the neurons for the respective layer $l-1$. $\boldsymbol{W} \in R^{n^{[l]} \times n^{[l-1]}}$ is the weight matrix, $\boldsymbol{b} \in R^{n^{[l]} \times 1}$ is the corresponding bias vector. Written-out for each element, the equation changes to:

$$z_i^{[l]} = \sum_{j=1}^{n^{[l-1]}} o_j^{[l-1]} \cdot w_{ij}^{[l]} + b_{ij}^{[l]} \text{with } i = \{1, .., n^{[l]}\} \tag{19}$$

Each element of the output vector $\boldsymbol{o}^{[l]}$ is defined as a pre-defined activation function $a(\cdot)$ applied to the corresponding linear vectors $\boldsymbol{z}^{[l]}$, element-wise:

$$\boldsymbol{o}^{[l]} = a(\boldsymbol{z}^{[l]}) \tag{20}$$

Written-out, the expression for each element is:

$$o_i^{[l]} = a(z_i^{[l]}), \text{ with } i = \{1, .., n^{[l]}\} \tag{21}$$

In the case of the input layer $l = 0$ and the output layer $l = L$ the output vector $\boldsymbol{o}$ is the input vector $\boldsymbol{x}$ and the output vector $\hat{\boldsymbol{y}}$, respectively:

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{n^{[0]}} \end{pmatrix} = \boldsymbol{o}^{[0]}, \text{ with } l = 0 \tag{22}$$

$$\hat{\boldsymbol{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_{n^{[L]}} \end{pmatrix} = \boldsymbol{o}^{[L]}, \text{ with } l = L \tag{23}$$

During the training process a neural network compares its own produced response vector $\hat{\boldsymbol{y}}$ with the true response vector $\boldsymbol{y}$ with the help of a loss function $\mathcal{L}$. Over the training process, the neural network model ideally converges this loss towards zero and its loss serves as a performance metric of the model's accuracy to approximate the training data. The loss function $\mathcal{L}$ is defined as the sum of all losses, which is simply the response loss $\mathcal{E}_{\text{Response}}$:

$$\mathcal{L} = \mathcal{E}_{\text{Response}} \tag{24}$$

The response loss $\mathcal{E}_{\text{Response}}$ is defined as the half-mean squared error of the model's response $\hat{\boldsymbol{y}}$ to the true response $\boldsymbol{y}$, whereby $\|\cdot\|$ denotes the Euclidean vector norm:

$$\mathcal{E}_{\text{Response}} = \frac{1}{2} \cdot \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2 = \sum_{i=1}^{n^{[L]}} \frac{1}{2} \cdot (\hat{y}_i - y_i)^2 \tag{25}$$

3) BACKPROPAGATION

For the training process, it is necessary to optimise the model parameters, referred to as θ, consisting of the weights and biases of the neural network. The neural network needs to identify the gradient of the loss with respect to each model parameter $\frac{\partial \mathcal{L}}{\partial \theta}$, for the minimization of the loss $\mathcal{L}$. To compute all gradients for the model parameter update step, the backpropagation follows after the forward pass. The forward pass populates each parameter of the neural network model and gives the current model response $\hat{\boldsymbol{y}}$. θ includes both weights w or biases b of its respective layer. To compute the gradients with respect to the model parameters of layer $l = p$, automatic differentiation is used, which applies the chain rule during backpropagation to evaluate each partial derivative step by step. The general formulation for a neural network with a single neuron in each of its layers defines the backpropagation for the partial derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^{[p]}}$ of the loss $\mathcal{L}$ with respect to the model parameters $\boldsymbol{\theta}^{[p]}$ per chain rule as:

$$\frac{\partial \mathcal{L}}{\partial \theta^{[p]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o^{[L]}} \frac{\partial o^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial o^{[L-1]}} \cdots \frac{\partial o^{[p]}}{\partial z^{[p]}} \frac{\partial z^{[p]}}{\partial \theta^{[p]}} \quad \text{with } 1 \leq p \leq L \tag{26}$$

Each partial derivative can then be evaluated following these fundamental derivatives. For $\frac{\partial \mathcal{L}}{\partial \hat{y}}$ and $\frac{\partial \hat{y}}{\partial o^{[L]}}$:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y \text{ and } \frac{\partial \hat{y}}{\partial o^{[L]}} = 1 \tag{27}$$

And the repeated partial derivatives through the layers:

$$\frac{\partial o^{[l]}}{\partial z^{[l]}} = a'(z^{[l]}) \text{ and } \frac{\partial z^{[l]}}{\partial o^{[l-1]}} = w^{[l]} \tag{28}$$

And for target layer $l = p$ we have:

$$\frac{\partial z^{[p]}}{\partial \theta^{[p]}} = \begin{cases} o^{[p-1]} & \text{if } \theta^{[p]} = w^{[p]} \\ 1 & \text{if } \theta^{[p]} = b^{[p]} \end{cases} \tag{29}$$

whereas in the last equation the distinction occurs on whether the model parameter θ is a weight w or bias b. When considering the order of operation in a feedforward neural network, each partial derivative with respect to model parameters of layer p is preceded by the partial derivative with respect to model parameters of its following layer $p+1$. Next, we define the partial derivative with matrices and vectors, since we are dealing with more than a single neuron in the layers. With respect to layer l we have the partial derivative with respect to the non-linear output $\boldsymbol{\delta}^{[l]}$:

$$\boldsymbol{\delta}^{[l]} = \frac{\partial \boldsymbol{z}^{[l+1]}}{\partial \boldsymbol{o}^{[l]}} \otimes \boldsymbol{\delta}^{[l+1]} \odot \frac{\partial \boldsymbol{o}^{[l]}}{\partial \boldsymbol{z}^{[l]}} \tag{30}$$

The operator $\odot$ denotes the Hadamard product, which is an element-wise multiplication, whereas the operator $\otimes$ denotes basic matrix multiplication. The terms are defined as follows:

$$\frac{\partial \boldsymbol{z}^{[l+1]}}{\partial \boldsymbol{o}^{[l]}} = (\boldsymbol{W}^{[l+1]})^T \in R^{n^{[l]} \times n^{[l+1]}} \tag{31}$$

$$\frac{\partial \boldsymbol{o}^{[l]}}{\partial \boldsymbol{z}^{[l]}} = a'(\boldsymbol{z}^{[l]}) \in R^{n^{[l]} \times 1} \tag{32}$$

The $\boldsymbol{\delta}^{[L]}$ at the last layer L is defined as:

$$\boldsymbol{\delta}^{[L]} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \odot \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{[L]}} \in R^{n^{[L]} \times 1} \tag{33}$$

From here on, we can define the partial derivatives with respect to model parameters in the following:

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \boldsymbol{\delta}^{[l]} \otimes \frac{\partial \boldsymbol{z}^{[l]}}{\partial \boldsymbol{W}^{[l]}} \in R^{n^{[l]} \times n^{[l-1]}} \tag{34}$$

In this term, we have defined $\frac{\partial \boldsymbol{z}^{[l-1]}}{\partial W^{[l]}}$ as:

$$\frac{\partial \boldsymbol{z}^{[l]}}{\partial \boldsymbol{W}^{[l]}} = (\boldsymbol{o}^{[l-1]})^T \tag{35}$$

For the biases, we have instead:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{[l]}} = \boldsymbol{\delta}^{[l]} \in R^{n^{[l]} \times 1} \tag{36}$$

From here on we define gradient vectors $\nabla \mathcal{L}^{[l]}$ for each layer containing all partial derivatives in a single column by vectorization $\text{vec}(\cdot)$ of their partial derivative matrix, which has the same dimensions as the respective weight matrix and bias vector.

$$\text{vec}(\boldsymbol{A}) = \begin{pmatrix} \text{vec}(\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{[l]}}) & \text{vec}(\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{[1]}}) \end{pmatrix}^T \tag{37}$$

We define a gradient vector for each weight matrix of layer l as:

$$\nabla \mathcal{L}^{[l]} = \begin{pmatrix} \text{vec}(\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{[l]}}) & \text{vec}(\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{[1]}}) \end{pmatrix}^T \tag{38}$$

Then, we concatenate all gradient vectors for use in residual weighting methods mentioned in later sections, which gives us the total gradient vector:

$$\nabla \mathcal{L} = \begin{pmatrix} \nabla \mathcal{L}^{[1]} & \nabla \mathcal{L}^{[2]} & \dots & \nabla \mathcal{L}^{[L]} \end{pmatrix}^T \tag{39}$$

III. DEVELOPMENTS

A. GENERAL REMARKS

The inclusion of sensitivities during training of neural networks provides performance improvement in the form of improved loss convergence, lower final loss and higher accuracy on validation data. When having data provided by a numerical simulation such as FEM, which is capable of also providing sensitivities w.r.t. design parameters through adjoint or direct methods, Sobolev training is a quick and worthwhile adjustment. Further in this section, Sobolev training will be expanded on, to provide an expansion of the basic ANN theory in the earlier section. When considering this Sobolev training, a weighting issue arises, as the neural network training has to consider responses as well as sensitivities in its algorithm. In regards to this issue, the first step is to find the optimal weights of said objectives for the optimal convergence performance. This however begs the question of how to decide which of the responses and sensitivities are in how far important for the convergence performance and ultimately the accuracy of the final trained model.

B. SOBOLEV TRAINING

Sobolev trained neural networks follow the general structure of a basic ANN. Instead of adding additional outputs by increasing the output layer size, the target loss function used for the optimization through gradient descent methods is adjusted to consider the additional sensitivities with respect to the inputs. The equations will show the similarity of terms of partial derivatives inside the neural network, explaining why Sobolev training is computationally efficient, when these sensitivities are available.

1) PREPROCESSING

For the preprocessing, to obtain the sensitivity of the first standardised response to a standardised input, $\frac{\partial y_{1,s}}{\partial x_{e,s}}$, the corresponding standard deviations $\sigma_{x_e}, \sigma_{y_1}$ are applied to the sensitivity $\frac{\partial y_1}{\partial x_e}$ as follows:

$$\frac{\partial y_{1,s}}{\partial x_{e,s}} = \frac{\partial y_1}{\partial x_e} \cdot \frac{\sigma_{x_e}}{\sigma_{y_1}} \tag{40}$$

This is done for each element e of $\boldsymbol{x}$ and for every element of $\boldsymbol{y}$. Going forward, we leave out subscripts d, e, s for simplicity.

2) SENSITIVITIES

When sensitivities of a system are available, especially when their additional calculation is computationally cheap, the implementation of sensitivities during training can effectively be considered. The main goal of applying sensitivities for model training is to compare the sensitivity of the model response defined by its model parameters to the true sensitivity of the true response. As a general chain rule expression, again for a neural network with only a single neuron in each of its layers, the sensitivity of the model $\frac{\partial \hat{y}}{\partial x}$ is defined for input x and output $\hat{y}$:

$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial \hat{y}}{\partial o^{[L]}} \frac{\partial o^{[L]}}{\partial z^{[L}} \frac{\partial z^{[L]}}{\partial o^{[L-1]}} \cdots \frac{\partial o^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial x} \tag{41}$$

With multiple inputs and outputs, we have the Jacobian $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}} \in R^{n^{[L]} \times n^{[0]}}$ which is defined as:

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial x_1} & \cdots & \frac{\partial \hat{y}_1}{\partial x_{n^{[0]}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_{n^{[L]}}}{\partial x_1} & \cdots & \frac{\partial \hat{y}_{n^{[L]}}}{\partial x_{n^{[0]}}} \end{bmatrix} \tag{42}$$

When considering multiple neurons, we again introduce matrices and vectors per earlier definition. However, we redefine $\boldsymbol{\delta}^{[L]}$, so as not to include the derivation of the loss function:

$$\boldsymbol{\delta}^{[L]} = \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{[L]}} \in R^{n^{[L]} \times 1} \tag{43}$$

We can now easily define the partial derivative with respect to inputs as follows:

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}} = (\boldsymbol{W}^{[1]})^T \otimes \boldsymbol{\delta}^{[1]} \tag{44}$$

This contains all sensitivities of the neural network model. All the partial derivatives for loss gradient calculations are already evaluated during the forward pass and backpropagation of the basic ANN. This allows efficient computation of sensitivity related expressions without increasing training time comparatively to the basic neural network, emphasising the performance improvement through Sobolev training. The new loss $\mathcal{L}$ is now defined as the sum of the response losses $\mathcal{E}_{\text{Response}}$ and the sensitivity losses $\mathcal{E}_{\text{Sensitivity}}$:

$$\mathcal{L} = \mathcal{E}_{\text{Response}} + \mathcal{E}_{\text{Sensitivity}} \tag{45}$$

Similar to the response losses $\mathcal{E}_{\text{Response}}$, the sensitivity losses $\mathcal{E}_{\text{Sensitivity}}$ are expressed as the half-mean squared error of the computed model's sensitivity and the true sensitivities. For each input, the respective sensitivity loss is computed with the Euclidean vector norm of its vectors $\frac{\partial \hat{\boldsymbol{y}}}{\partial x_j}$ and $\frac{\partial \boldsymbol{y}}{\partial x_j}$:

$$\begin{aligned} \mathcal{E}_{\text{Sensitivity}} &= \frac{1}{2} \cdot \sum_{j=1}^{n^{[0]}} \| \frac{\partial \hat{\boldsymbol{y}}}{\partial x_j} - \frac{\partial \boldsymbol{y}}{\partial x_j} \|^2 \\ &= \frac{1}{2} \cdot \sum_{i=1}^{n^{[L]}} \sum_{j=1}^{n^{[0]}} \left(\frac{\partial \hat{y}_i}{\partial x_j} - \frac{\partial y_i}{\partial x_j} \right)^2 \end{aligned} \tag{46}$$

The model training occurs unchanged, by solving the minimization of its defined loss $\mathcal{L}$ for its model parameters $\boldsymbol{\theta}$:

$$\begin{aligned} \boldsymbol{\theta} &= \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\ \text{with } \boldsymbol{\theta} &= \{\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\} \; for \; l \in [1, L] \end{aligned} \tag{47}$$

While the sensitivities are expanding the loss function, they are not an output of additional neurons in the output layer. The Sobolev training operates akin to a loss regularization term.

3) BACKPROPAGATION

Unchanged to the basic ANN, the process of computing the gradients with respect to the model parameters of layer $l = p$ is done by using automatic differentiation. The chain rule is applied during backpropagation to evaluate each partial derivative step by step numerically. The general equation for partial derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^{[p]}}$ of the loss $\mathcal{L}$ with respect to model parameters $\boldsymbol{\theta}^{[p]}$ follows, whereas an additional addend expands the previous equation, that is the partial derivative of the loss along the sensitivities, $\frac{\partial \mathcal{L}}{\partial \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}} \frac{\partial \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}}{\partial \theta^{[p]}}$:

$$\frac{\partial \mathcal{L}}{\partial \theta^{[p]}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{o}^{[L]}} \frac{\partial \boldsymbol{o}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial \boldsymbol{z}^{[L]}}{\partial \boldsymbol{o}^{[L-1]}} \cdots \frac{\partial \boldsymbol{o}^{[p]}}{\partial \boldsymbol{z}^{[p]}} \frac{\partial \boldsymbol{z}^{[p]}}{\partial \theta^{[p]}} + \frac{\partial \mathcal{L}}{\partial \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}} \frac{\partial \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}}{\partial \theta^{[p]}} \quad \textit{with} \quad 1 \leq p \leq L \tag{48}$$

For $\frac{\partial \mathcal{L}}{\partial \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}}$, the evaluation is simply the difference between the neural networks sensitivity and the true sensitivity:

$$\frac{\partial \mathcal{L}}{\partial \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}} = \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}} - \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \tag{49}$$

As previously shown, the sensitivity of the model response is also expressed per chain rule. Therefore, for layer $l = p$, it simply follows for $\frac{\partial}{\partial \theta^{[p]}} \cdot \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)$:

$$\frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)}{\partial \theta^{[p]}} = \frac{\partial}{\partial \theta^{[p]}} \cdot \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{o}^{[L]}} \frac{\partial \boldsymbol{o}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial \boldsymbol{z}^{[L]}}{\partial \boldsymbol{o}^{[L-1]}} \cdots \frac{\partial \boldsymbol{o}^{[1]}}{\partial \boldsymbol{z}^{[1]}} \frac{\partial \boldsymbol{z}^{[1]}}{\partial \boldsymbol{x}}\right) \tag{50}$$

For the partial derivative relating to the model sensitivity, that is the partial derivative with respect to input, for layer p this expression has to be differentiated for all layers $l \geq p$. This is because $\theta^{[p]}$ is nested inside all layers after layer p propagation-wise. Consider the last layer for our simple expressions of depth 1 in all layers, with the previous defined state of all parts of $\frac{\partial}{\partial \theta^{[p]}} \cdot \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)$, for the case $p = L$, this expression turns to:

$$\frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)}{\partial \theta^{[L]}} = \frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{o}^{[L]}} \frac{\partial \boldsymbol{o}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial \boldsymbol{z}^{[L]}}{\partial \boldsymbol{o}^{[L-1]}}\right)}{\partial \theta^{[L]}} \cdot \left(\cdots \frac{\partial \boldsymbol{o}^{[1]}}{\partial \boldsymbol{z}^{[1]}} \frac{\partial \boldsymbol{z}^{[1]}}{\partial \boldsymbol{x}}\right) \tag{51}$$

Since the goal is to compute the partial derivative with respect to $\theta^{[L]}$, any terms propagated before this model parameter are independent of it, allowing these independent terms to be factored out of the partial derivative with respect to $\theta^{[L]}$. The partial derivatives of $\frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{o}^{[L]}} \frac{\partial \boldsymbol{o}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial \boldsymbol{z}^{[L]}}{\partial \boldsymbol{o}^{[L-1]}}\right)}{\partial \theta^{[L]}}$ are left for differentiation, which were previously defined. Per product and chain rule, it follows:

$$\frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{o}^{[L]}} \frac{\partial \boldsymbol{o}^{[L]}}{\partial \boldsymbol{z}^{[L]}} \frac{\partial \boldsymbol{z}^{[L]}}{\partial \boldsymbol{o}^{[L-1]}}\right)}{\partial \theta^{[L]}} = \frac{\partial \left(1 \cdot a'(z^{[L]}) \cdot w^{[L]}\right)}{\partial \theta^{[L]}} \tag{52}$$

and for θ being a weight w and bias b, respectively:

$$\frac{\partial \left(1 \cdot a'(z^{[L]}) \cdot w^{[L]}\right)}{\partial \theta^{[L]}} = \begin{cases} a''(z^{[L]}) \cdot o^{[L-1]} \cdot w^{[L]} + a'(z^{[L]}) \\ \text{for } \theta^{[L]} = w^{[L]} \\ \\ a''(z^{[L]}) \cdot w^{[L]} \\ \text{for } \theta^{[L]} = b^{[L]} \end{cases} \tag{53}$$

Through these general equations, all gradients for the Sobolev trained neural network can be computed. In the case of a simple activation function, such as the rectified linear unit applied in this work, second-order derivatives of the activation function are 0, simplifying the gradient calculation. When considering multiple neurons, with the previous definition of $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}$ and our newly defined loss function, we also define the first partial derivative with respect to model parameters for the sensitivities:

$$\frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)}{\partial \boldsymbol{W}^{[1]}} = \boldsymbol{I}^{W^{[1]}} \otimes \boldsymbol{\delta}^{[1]} \tag{54}$$

With $\boldsymbol{I}^{W^{[1]}}$ being an Identity matrix with the same dimensions as its respective weight matrix of the first layer. For all other layers, we have:

$$\frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial \left(\boldsymbol{W}^{[1]} \otimes \boldsymbol{\delta}^{[1]}\right)}{\partial \boldsymbol{W}^{[l]}} \tag{55}$$

Following the chain rule, when reaching the respective layer $\boldsymbol{\delta}^{[l-1]}$, we obtain:

$$\frac{\partial \boldsymbol{\delta}^{[1-1]}}{\partial \boldsymbol{W}^{[l]}} = \boldsymbol{I}^{W^{[l]}} \otimes \boldsymbol{\delta}^{[l]} \odot \frac{\partial \boldsymbol{o}^{[l-1]}}{\partial \boldsymbol{z}^{[l-1]}} \tag{56}$$

Since we use the ReLU activation function, all partial derivatives per product rule of $\frac{\partial \boldsymbol{o}^{[l]}}{\partial \boldsymbol{z}^{[l]}}$ turn 0 and are left out of this equation. The gradients of the Sobolev loss are now defined as:

$$\nabla \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{W}^{[l]}} + \frac{\partial \mathcal{L}}{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)} \frac{\partial \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{x}}\right)}{\partial \boldsymbol{W}^{[l]}} \tag{57}$$

Once all gradients have been computed, the applied training algorithm adjusts each model parameter unchanged to the basic ANN by an update step which now considers the gradients w.r.t. model parameters for both responses as well as sensitivities.

C. RESIDUAL WEIGHTS

When considering the different losses of response and sensitivity, the optimization task has become more involved. Each individual loss represents its own optimization target for which convergence towards zero will improve accuracy of the neural network model. But since the actual target value of the model is to approximate the true output y only, it is still considered a single-objective optimization problem. Nevertheless, by introducing multiple new losses, which are

indirectly approximated, a new space of loss optimization can be considered. In this more involved optimization problem, there is an optimal weighting of response and sensitivity losses to obtain optimal training convergence. When considering the optimal ratio of the loss addends, there are a variety of approaches to consider. In this work, various methods to optimise the residual weights are considered and empirically evaluated. All optimization methods used for the residual weights of the losses will apply the same optimization algorithm, ADAM, which is applied to the model parameters of the neural network. The general goal of these approaches is, that during any training step, the defined sum of response and sensitivity losses arrange a traversable space by ratio of residual weights, where there should exist a ratio for optimal performance or convergence during its training.

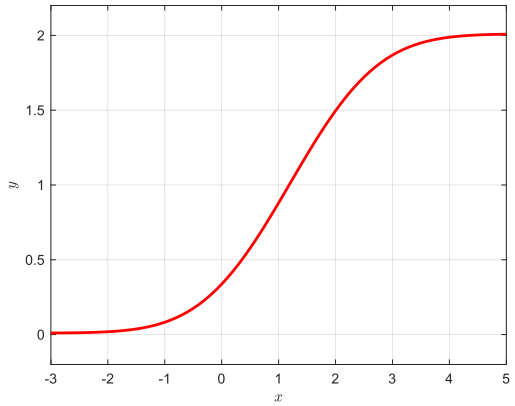

FIGURE 1. Residual weight function.

The methods are employed to find this ratio and thereby improve convergence. To make this possible, the previous loss $\mathcal{L}$ is expanded by residual weights $\boldsymbol{\lambda}_R$ and $\boldsymbol{\lambda}_S$:

$$\mathcal{L} = \boldsymbol{\lambda}_R \, \mathcal{E}_{\text{Response}} + \boldsymbol{\lambda}_S \, \mathcal{E}_{\text{Sensitivity}} \tag{58}$$

To limit the range of the residual weights $\boldsymbol{\lambda}_i$ and stabilise the training process, they are evaluated by a function. This limits the residual weights between the range $[\epsilon_0; 2+\epsilon_0]$ and $\epsilon_0 = 0.01$ is added to avoid elimination of response and sensitivity loss and numerical instability. Specifically, the initial residual weight $\boldsymbol{\lambda}_{i,prev}$ is applied to the error function erf$(\cdot)$, which returns $\boldsymbol{\lambda}_i$, which is then used for subsequent computation:

$$\boldsymbol{\lambda}_i = 1 - \text{erf}\left(\frac{1.2 - \boldsymbol{\lambda}_{i,prev}}{\sqrt{3}}\right) + \epsilon_0 \tag{59}$$

This is then repeated for during every training step to reevaluate the residual weights. While previously, the loss was minimized for model parameters $\boldsymbol{\theta}$, now the model training occurs by solving the defined loss $\mathcal{L}$ twofold. First, the loss $\mathcal{L}$ is minimized for its model parameters $\boldsymbol{\theta}$, just like with a regular neural network:

$$\begin{aligned} \boldsymbol{\theta} &= \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \\ \text{with } \boldsymbol{\theta} &= \{\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\} \quad \text{for } l \in [1, L] \end{aligned} \tag{60}$$

TABLE 1. Methods for optimising the residual weights $\boldsymbol{\lambda}$.

#	Method
1	$\arg\min_{\boldsymbol{\lambda}} \mathcal{L}$
2	$\arg\max_{\boldsymbol{\lambda}} \mathcal{L}$
3	$\arg\min_{\boldsymbol{\lambda}} \lvert\nabla\mathcal{L}\rvert$
4	$\arg\max_{\boldsymbol{\lambda}} \lvert\nabla\mathcal{L}\rvert$
5	$\arg\min_{\boldsymbol{\lambda}} Var(\mathcal{L})$
6	$\arg\min_{\boldsymbol{\lambda}} \left(1 - \frac{\nabla\mathcal{L}\cdot\nabla\mathcal{L}_i}{\lvert\nabla\mathcal{L}\rvert\cdot\lvert\nabla\mathcal{L}_i\rvert}\right)$
7	$\arg\min_{\boldsymbol{\lambda}} \left(\frac{\nabla\mathcal{L}\cdot\nabla\mathcal{L}_i}{\lvert\nabla\mathcal{L}\rvert\cdot\lvert\nabla\mathcal{L}_i\rvert}\right)^2$
8	$\arg\min_{\boldsymbol{\lambda}} \left(1 - \frac{\nabla\mathcal{L}_{\text{Response}}\cdot\nabla\mathcal{L}_j}{\lvert\nabla\mathcal{L}_{\text{Response}}\rvert\cdot\lvert\nabla\mathcal{L}_j\rvert}\right)$
9	$\arg\min_{\boldsymbol{\lambda}} \left(\frac{\nabla\mathcal{L}_{\text{Response}}\cdot\nabla\mathcal{L}_j}{\lvert\nabla\mathcal{L}_{\text{Response}}\rvert\cdot\lvert\nabla\mathcal{L}_j\rvert}\right)^2$

Lastly, the various methods listed in Table. 1 are used to update the residual weights $\boldsymbol{\lambda}_i$. These methods cover a variety of optimization targets during training. Certain methods only differ in their application of said optimization target $\mathcal{G}(\cdot)$:

$$\begin{aligned} \boldsymbol{\lambda} &= \arg\min_{\boldsymbol{\lambda}} \mathcal{G}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \\ \text{with } \boldsymbol{\lambda} &= [\epsilon_0; 2+\epsilon_0] \text{ for } \epsilon_0 = 0.01 \end{aligned} \tag{61}$$

Table. 1 covers all applied methods for the optimization of the residual weights $\boldsymbol{\lambda}$ inside the loss function. These methods only apply to the residual weight optimization, in parallel there is still the usual optimization of neural network weights and biases $\boldsymbol{\theta}$ per minimization of the MSE loss. In order they are 1, minimization and 2, maximization of the total loss function $\mathcal{L}$. These are the most basic options of target functions, whereby minimising the loss follows the same idea as the optimization of model parameters as outlined with mSANN. The maximization of loss w.r.t. residual weights $\boldsymbol{\lambda}$ in parallel to the general minimization of the loss w.r.t. model parameters θ should allow for the maximum possible gradient step correction. Arguably, it is also expected that minimization of the loss will simply converge the residual weights to 0 and the maximization of the loss will lead to increasing residual weights since losses are defined as > 0. Then we have for 3, minimization and 4, maximization of the magnitude of the total loss gradient $|\nabla\mathcal{L}|$. These target functions directly optimise in regards to the convergence rate $|\nabla\mathcal{L}|$. The idea and expected behaviors are similar to the methods 1 and 2. Next, for method 5, minimization of the variance $Var(\mathcal{L})$ between each individual loss, which aims to reduce the difference in performance between the individual losses. This should lead to all individual losses working in tandem, either all performing well or all performing bad. Methods 6 to 9 are various methods targeting the angle between different loss gradient vectors through the cosine similarity. Cosine losses or methods of targeting cosine similarity in neural networks are used in various works, most notably in image recognition, where similar feature vectors are compared [46], [47], [48]. In the case of 6 and 7, the target

is to align the gradient vector of each individual loss $\nabla\mathcal{L}_i$ to the total loss gradient vector $\nabla\mathcal{L}$ by minimising the cosine distance and the squared cosine similarity, respectively. Similarly, methods 8 and 9 apply the minimization of the cosine distance and the squared cosine similarity. However, in their case, the target is to align each individual loss gradient vector $\nabla\mathcal{L}_i$ with the gradient vector of the loss for the response $\nabla\mathcal{L}_R$ instead, since the response is the main target of the model. The cosine distance minimization should force equal angles between the respective gradient vectors. Similarly, the squared cosine distance minimization should force orthogonality between the respective gradient vectors.

TABLE 2. Methods with none or fixed residual weight optimization.

#	Method	$\lambda_{\text{InitialState}}$
10	SNN	$[1, 1, 1]$
11	Basic ANN	$[1, 0, 0]$
12	SNN Exp. Decay	$[1, 1, 1]$
13	SNN Exp. Increase	$[0, 0, 0]$

For comparison, the results will include additional model versions, see Table. 2, which omit an adaptive weighting scheme and the error function in (59). The last column shows the initialised residual weight values $\boldsymbol{\lambda}_{\text{InitialState}}$, the first value weighting the response loss, the second value weighting the sensitivity w.r.t. the first input variable and the third value weighting the sensitivity w.r.t the second input variable. Modes 10 and 11 are the Sobolev trained model and a basic ANN respectively, with the basic ANN not considering sensitivities and therefore setting the respective weights to zero. In SNN the weights are fixed at one each, in order to have a baseline Sobolev for comparison, which we seek to improve. There is also modes 12 and 13, which add exponential decay, see (62), and exponential increase, see (63), of the residual weight value. For this two modes, the residual weights $\boldsymbol{\lambda}_i$ are adjusted each epoch by the rate μ, the current number of epochs i_{Epochs} and the residual weight of the previous epoch $\boldsymbol{\lambda}_{i,previous}$.

$$\boldsymbol{\lambda}_i = \frac{\boldsymbol{\lambda}_{i,previous}}{1 + (\mu \cdot i_{Epochs})} \tag{62}$$

$$\boldsymbol{\lambda}_i = \boldsymbol{\lambda}_{i,previous} \cdot (1 + (\mu \cdot i_{Epochs})) \tag{63}$$

IV. TOOLS AND APPLICATION

A. GENERAL REMARKS

For the realization of this work, a FEM simulation was used to provide the necessary training data to train multiple neural network models. In addition, the following section will go into the details of each tool used.

B. FINITE ELEMENT MODEL

All neural network models approximate the data generated by a finite element model. The FEM evaluates a two dimensional mechanical system of linear and non linear elasticity. In this example a simple hook, see Fig. 2, is attached to a wall. Further, a force is applied on top of the hook, to emulate an object being attached to the hook. The shape of this hook is then modified through two design parameters.

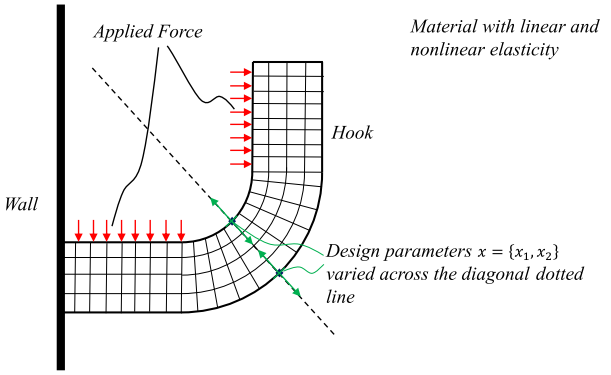

FIGURE 2. Sketch of the hook system approximated by the finite element model. The design parameters x_1, x_2 in the range $[-1, 1]$ adjust the blue marked mesh nodes along the green axis.

Their values shift two controls points of an underlying geometry description in the midsection of the hook orthogonal to the hook radius, thereby reshaping the hook, see control points 9 and 10 marked in Fig. 3. With these two design variables, the FEM captures the geometric uncertainty of the system, which is crucial in metal forming with for example springback behavior. The finite element model evaluates this mechanical system in terms of its stress state for an array of design parameter pairs.

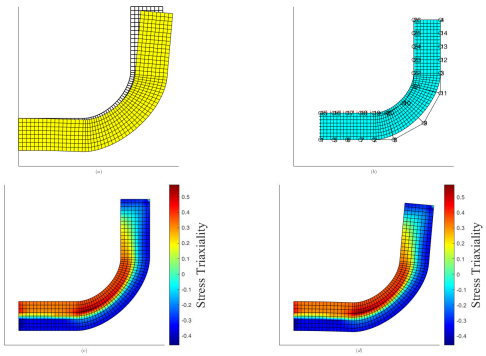

FIGURE 3. Visualization of the hook mesh deformation (a), the model's control points (b) and stress state undeformed (c) and deformed (d) for design parameters $x_1, x_2 = (0, 0)$.

The design parameters are evaluated in the range $[-1, 1]$ to stay in the range of physically plausible deformation of the hook geometry. These responses and sensitivities are then used for training the neural network models. The focus of this paper is the stress triaxiality of the material. The stress triaxiality specifically is a desired final quantity as it serves as a general indicator in regards to the material damage of an object such as void fraction area. More precisely, the stress triaxiality of a single specific node is chosen, where the finite element model evaluation produces the most critical

value. For the non linear elasticity evaluation, the material is specified as St. Venant-Kirchoff material and the finite element model evaluates the hook in the plane stress state. For a general visualization of the use case, see Fig. 3.

C. NEURAL NETWORK MODELS

In this paper, all neural network models are trained in the MATLAB R2023b software environment. To train the models, the MATLAB Deep Learning Toolbox is utilised. Since the goal is to compare different residual weighting methods, the neural network design is simplified. For the choice of model architecture, preliminary test runs were done in order to gauge feasible options. The primary goal of these preliminary test runs was to identify a model architecture that allowed for fast training time as well as a small approximation error without overfitting across the various methods applied in this paper. To allow the models to learn more complex patterns from the inputs, the hidden layers begin largest. All the following hidden layers reduce in size up to the output layer so that the learned features are reduced in dimensions and thereby simplified for the final output. Thus, the models are designed in small dimensions, with the hidden layers being made up of a "5-3-3" neuron-layer structure, see Fig. 4.

TABLE 3. Neural network hyperparameters.

Hyperparameter	Value
Minibatches	64
Learnrate	0.001
Grad Decay	0.9
Squared Grad Decay	0.999
Epsilon	10^{-8}

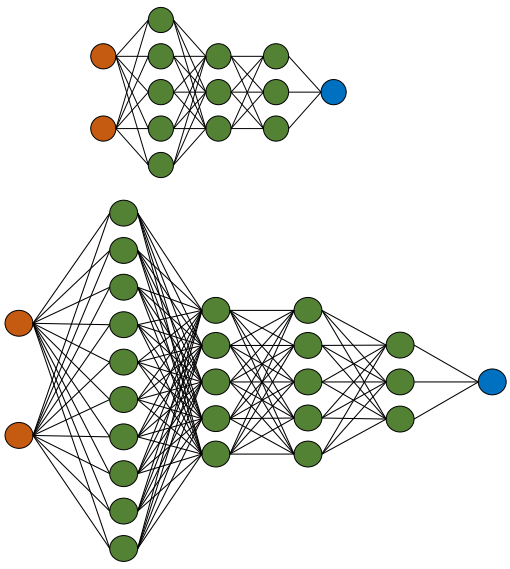

FIGURE 4. Neural network model layer and neuron structure with 2 Inputs and 1 Output. Top: Basic layer structure 5-3-3, Bottom: Additional layer structure 10-5-5-3.

Each model has two input neurons and a single output neuron. Each layer applies the rectified linear unit as an activation function. A second model layer structure is used later to analyse the effect of layer structure on training convergence. The second structure will have an additional hidden layer and contain more neurons, see Fig. 4. Furthermore, the training occurs in minibatches using the ADAM training algorithm. Each ANN model is trained on the same dataset and each result is the average of 100 trained models with random initialization of model parameters, to reduce bias from various factors.

The model uses 320 training data points in addition to 305 validation data points, whereby 625 data points are generated by the FEM. The 625 data points are evaluated in a range of design parameters of constant step size. For the validation data, data points of uniform step size are extracted. For each iteration, the current model state is used to calculate the relative l_2 error through use of the validation data. Table. 3 covers the important hyperparameters used to train the neural network models with the ADAM algorithm. The first results are produced after only 500 epochs of training to gauge the effectiveness of each method. Afterwards, additional results of the two best converging methods as well as the SNN are produced, which entails an expanded training for 1000 epochs. There are also additional results for 2000 epochs.

V. RESULTS

The results consist of multiple instances of the aforementioned models with varying training parameters. While there are multiple metrics that can be observed, the focus will be on the l_2 error concerning the validation data. The reason for this choice is multi-fold. First, the l_2 error as a metric allows validation of the trained model and an overview of overfitting and underfitting. Secondly, since the goal of this paper is to compare performance of various weighting methods with each other, the l_2 error is more indicative of the models approximation accuracy than the total loss. Since the models use Sobolev training and adjust the magnitude of the individual loss addends, the loss value of the different models does not provide meaningful information for comparison. Consider that in this paper the total loss is the sum of individual loss terms, which are scaled by their respective weights that are adjusted during training. Because of this the total loss is artificially scaled, making comparisons difficult. Instead, as will be seen later in this section, the individual losses without the weight scaling will be used for comparison. The information is visualised through various tables and graphs.

Furthermore, out of all the trained models for each mode, the best performing models are compared via box plots, with a secondary focus on the best performing residual weighting methods. The box plots visualise the average value over 100 trained models with a red line, with the inner 50% of values of the trained models inside of the blue box. The dotted black lines represent the rest of the data of the trained models with lower and upper ends. Finally, for the best performing

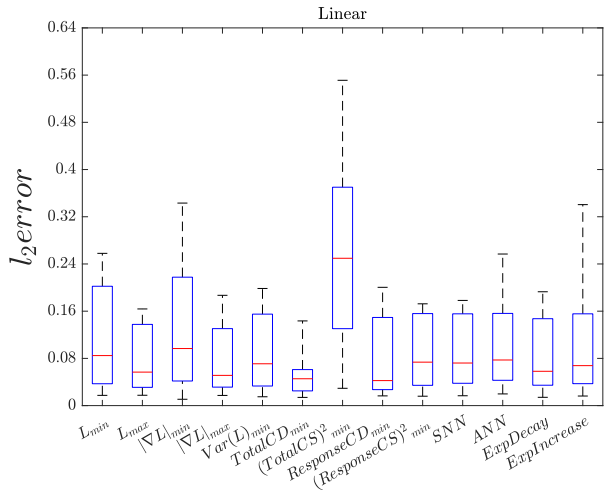

FIGURE 5. L_2 error spread of all training runs for each mode.

residual weighting methods, the 1000 epoch training runs are visualised with their loss and l_2 error curves over the iterations. On top of this, the individual losses of responses and sensitivities, as well as the residual weights, are plotted, to analyse for possible patterns. There are additional results for the purpose of analysing the effects of layers, epochs and activation functions, etc. on the training convergence of the models.

A. LINEAR CASE

For the linear elasticity case, training the models for 500 epochs results in the box plots in Fig. 5 for the l_2 error values. For better analysis, the l_2 error box-plots of residual weighting mode 2 and 6 are represented together with mode 9 for comparison in Fig. 6.

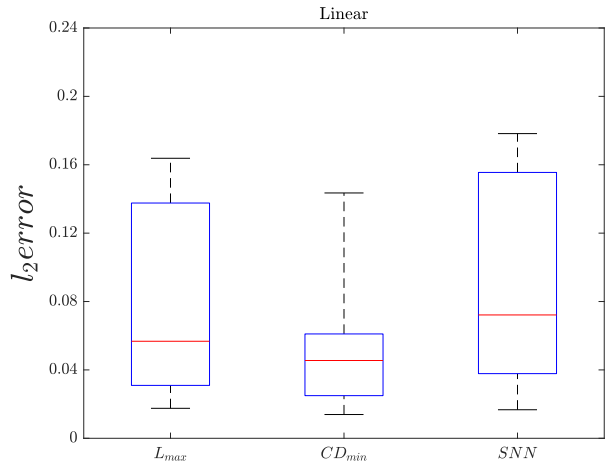

FIGURE 6. L_2 error spread of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included.

In addition, the averaged l_2 error and training time results of the multiple training runs for each mode are listed in Table. 4.

Following these results, modes 2 and 6 are chosen to be trained with 1000 epochs. While mode 8 also shows promising results, the optimization method is similar to the better performing mode 6 and is therefore not continued for 1000 epochs. In addition, the basic Sobolev model, mode 9, is trained with 1000 epochs for comparison. This results in the loss and l_2 error curves over the iterations are visualised in Fig. 7.

For further patterns, the individual losses of response and the two sensitivities are plotted in Fig. 8 and Fig. 9. It is to note that these individual losses do not include the residual weights, which is why the mode without residual weight optimization still has the lowest loss in Fig. 7.

Lastly, the final residual weight values themselves are plotted for comparison in Fig. 10.

TABLE 4. Average final l_2 error for each mode. The averaged training time is included as well.

Mode	L_2 Error	Training Time [s]
1	0.111	78
2	0.076	78
3	0.115	125
4	0.071	126
5	0.085	83
6	0.057	198
7	0.256	199
8	0.073	160
9	0.093	161
10	0.086	68
11	0.094	68
12	0.078	68
13	0.085	68

TABLE 5. Average final l_2 error for each mode. The averaged training time is included as well.

Mode	L_2 Error	Training Time [s]
1	0.115	78
2	0.088	78
3	0.134	124
4	0.093	123
5	0.09	80
6	0.087	191
7	0.213	197
8	0.091	144
9	0.092	144
10	0.089	67
11	0.112	68
12	0.103	67
13	0.105	67

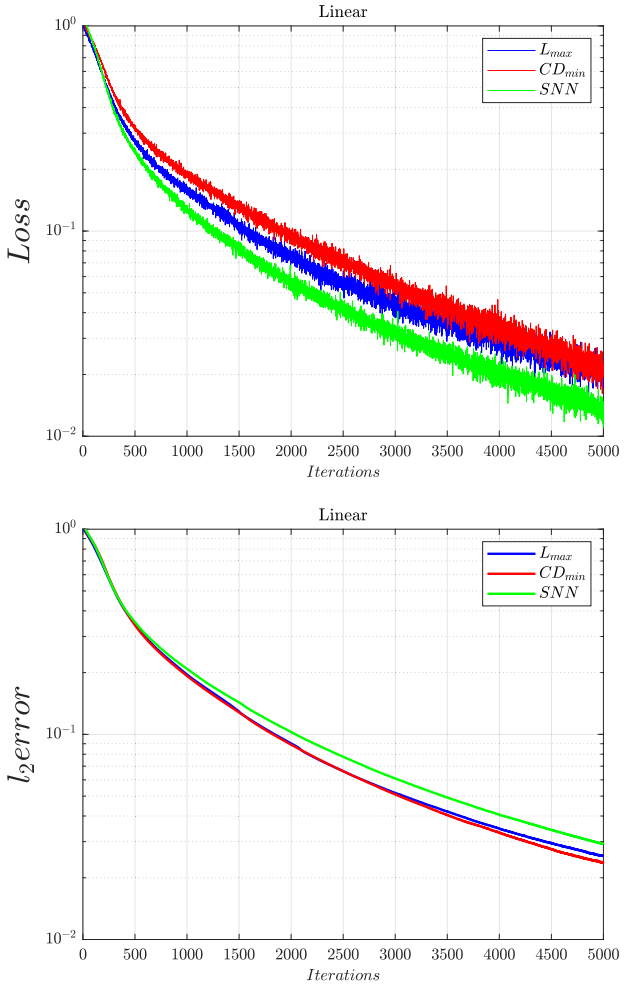

FIGURE 7. ***Top*: Loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included; Bottom: L_2 error over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included.**

B. NON-LINEAR CASE

For the non-linear elasticity case, training the models for 500 epochs results in the box plots in Fig. 11 for the l_2 error values.

For better analysis, the l_2 error box-plots of mode 2 and 6 are represented together with mode 9 for comparison, see Fig. 12.

In addition the averaged l_2 error and training time results for each method are listed in Table. 5.

Following these results, modes 2 and 6 are chosen to be trained with 1000 epochs. In addition, the basic Sobolev model, mode 9, is trained with 1000 epochs for comparison. This results in loss and l_2 error curves over the iterations in Fig. 13.

For further patterns, the individual losses of response and the two sensitivities are plotted in Fig. 14 and Fig. 15. It is to note that these individual losses do not include the residual weights, which is why the mode without residual weight optimization has the lowest loss in Fig. 13.

Lastly, the final residual weight values themselves are plotted for comparison in Fig. 16.

For closing, results for these same models, trained on 2000 epochs, are produced to observe the effects of the epoch length on the training convergence.

The Table. 6 shows the values for 500, 1000 and 2000 epochs. Furthermore, the effects of the model layer size is observed with additional results for a layer structure of 10-5-5-3 and compared to the original 5-3-3 model in Fig. 17 and Fig. 18.

Lastly, Table. 7 illustrates the difference of the average and minimum l_2 error for both layer structures.

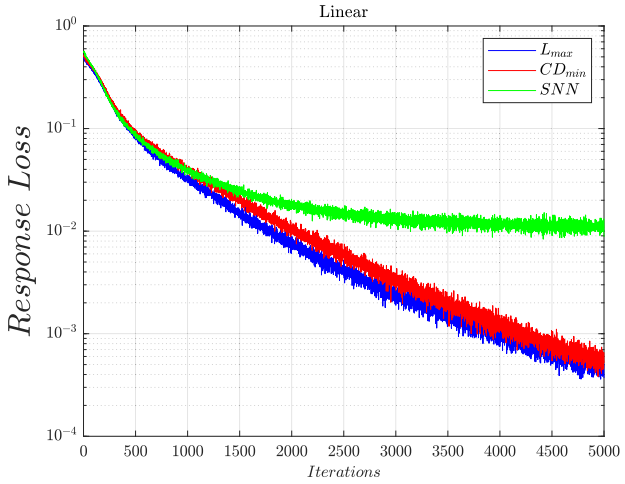

FIGURE 8. **Response loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included.**

TABLE 6. **Average and lowest l_2 error for *max* $\mathcal{L}$, *min* CD and *SNN* for different number of epochs.**

Mode	Epochs	Avg. L_2 Error [s]	Smallest L_2 Error
max $\mathcal{L}$	500	0.088	0.027
min $\boldsymbol{CD}$	500	0.087	0.028
SNN	500	0.089	0.025
max $\mathcal{L}$	1000	0.039	0.015
min $\boldsymbol{CD}$	1000	0.036	0.016
SNN	1000	0.045	0.016
max $\mathcal{L}$	2000	0.026	0.014
min $\boldsymbol{CD}$	2000	0.026	0.014
SNN	2000	0.025	0.013

VI. ANALYSIS AND DISCUSSION

While the analysis will be split into linear and non-linear case, the non-linear case will have a slightly expanded analysis. This is because when observing the first results in the non-linear case, the differences between the best performing modes is less pronounced, as will be discussed later. This is not the case for the linear results. The non-linear case discussion therefore also covers the 2000 epochs training results. In addition the effect of the expansion of layers is only

TABLE 7. Average and lowest l_2 error for *max* $\mathcal{L}$ and *min* CD for the two different layer sizes.

Mode	Layers	Avg. L_2 Error	Lowest L_2 Error
max $\mathcal{L}$	$5-3-3$	0.035	0.016
min $\boldsymbol{CD}$	$5-3-3$	0.034	0.016
max $\mathcal{L}$	$10-5-5-3$	0.022	0.011
min $\boldsymbol{CD}$	$10-5-5-3$	0.020	0.009

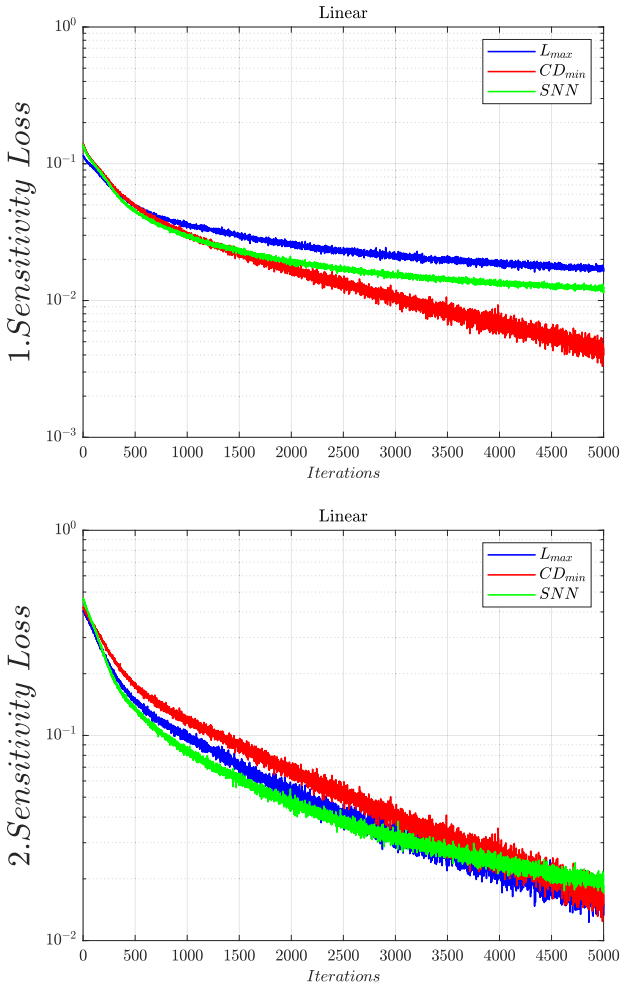

FIGURE 9. Top: First sensitivity loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included; Bottom: Second sensitivity loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included.

analysed for the non-linear case. It should be noted that when comparing FEM with any of the neural network models, in the case of more complex systems, such as in the non-linear case, the FEM is computationally more expensive with increasing data sets being generated. For example on the same hardware, when generating only a small set of 10^4 data points, the FEM takes up to ∼2200 seconds, where as the neural network models can be trained on only 10^2 data points to then generate the same 10^4 data points in under 1 second with satisfying accuracy. Especially when large amounts of data have to be produced to be used in, for example, evaluation of failure probabilities. This entails generating large datasets. Since the FEMs data generation is fixed per data point, this effectively linearly scales the required computation cost. In contrast, the neural network surrogates only need a preliminary training data set produced by either measurements or in this case FEM generation as a solid base to be trained on. Once the network is trained, large datasets can be generated at negligible numerical cost. This highlights the advantages of the proposed scheme in comparison to repeated FEM analysis.

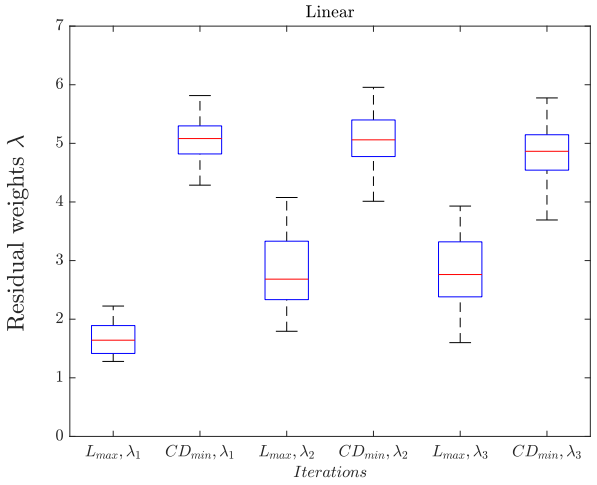

FIGURE 10. Spread of the final residual weight values of all training runs for *max* $\mathcal{L}$ and *min* CD.

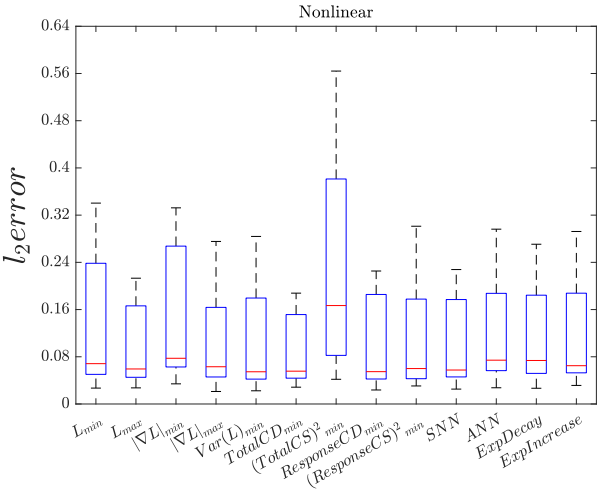

FIGURE 11. L_2 error spread of all training runs for each mode.

A. LINEAR CASE

The linear case results show decisively, which modes performs best. Fig. 5 shows the spread of the multiple runs of the different applied modes of training. It can be observed that mode 6 performs best. Following this, modes 2 and 4 perform second best. The other modes seem to perform

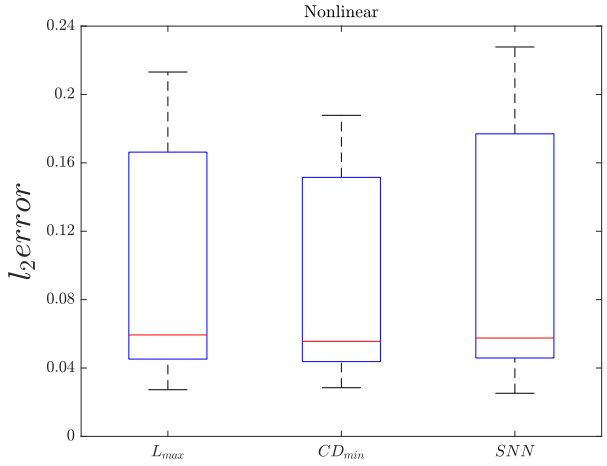

FIGURE 12. L_2 **error spread of** ***max*** $\mathcal{L}$ **and** ***min*** CD**. For comparison the mode without residual weight optimization is included.**

somewhat equal, with modes 1, 3 and 7 performing the worst. In Fig. 6, the difference in performance of mode 2 and 6, as well as the base performance of the Sobolev trained model, mode 10, becomes clearer. The gradient alignment applied in mode 6 shifts the spread of the l_2 error slightly further down but also increases the precision of the spread of values in comparison to mode 2 and 10. This can also be observed in the average being lower than the other two modes, as well as the lowest obtained value being better. In Table. 4 the difference of performance is visualised with the average scalar l_2 error values. First, mode 6 is clearly the best performing mode, followed by mode 4 and 2. However when considering the average training time for the modes, mode 6 is second to last. In comparison, mode 2 takes less than half the time and mode 4 only takes ∼60% to train. While the loss itself is no longer a good metric for comparison of modes, the modes 2 and 6 have a bigger loss than the basic Sobolev training, see Fig. 7. To actually judge the performance difference, Fig. 7 provides the l_2 error, where both modes outperform the basic Sobolev training mode, meaning it takes less time to train a model for a certain accuracy. It can also be observed that modes 2 and 6 perform relatively equal until iteration ∼2000, where mode 6 begins to outperform mode 2 by a slight margin. Both of these figures are the results of 1000 epoch training. Compared to the 500 epoch training, there is a performance difference in magnitude. However, when it comes to modes 2 and 6, which were performing the best, the difference in performance is unchanged. Interestingly, when observing the results in Fig. 8 and Fig. 9, mode 6 reduces the losses of all individual loss addends the best. Mode 2 seems to have difficulties approximating the first sensitivity and the basic Sobolev training mode only keeps up the performance for the second sensitivity. Still, for the total loss, the residual weights are included in the calculation, which leads to the basic Sobolev training mode having the lowest loss of all three, as mentioned for Fig. 7. When analysing the values the residual weights end up with, it can be seen in Fig. 10 that mode 2 generally converges the residual weights to lower ranges than mode 6. Additionally, while all three residual weights of mode 6 range in similar regions, in the case of mode 2, the residual weight of the response loss is generally in a lower range than the residual weights of the sensitivity losses. In both cases, the residual weight of the response loss seems to have the highest precision.

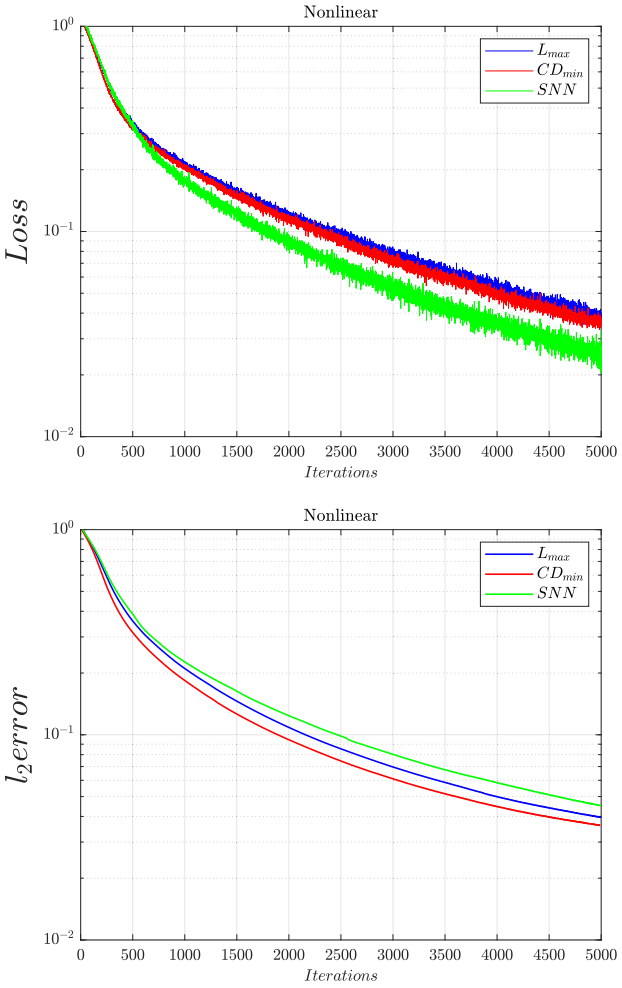

FIGURE 13. Top: Loss over the training iterations of ***max*** $\mathcal{L}$ **and** ***min*** CD**. For comparison the mode without residual weight optimization is included; Bottom:** L_2 **error over the training iterations of** ***max*** $\mathcal{L}$ **and** ***min*** CD**. For comparison the mode without residual weight optimization is included.**

B. NON-LINEAR CASE

The non-linear case results show how the performance between modes is not as clear as the linear case. Fig. 11 shows the spread of the multiple runs of the different applied modes of training. It can be observed that for the non-linear case mode 6 performs best as well. Following this, modes 2 and 4 perform second best. The other modes seem to perform somewhat equal, with modes 1, 3 and 7 performing the worst. However the difference in performance is not quite as big as it was in the linear case. Modes 2, 4 and 6 only outperform slightly. In Fig. 12, the difference in performance of mode 2 and 6, as well as the base performance of the Sobolev trained model, mode 10, becomes clearer. The gradient alignment applied in mode 6 increases the precision

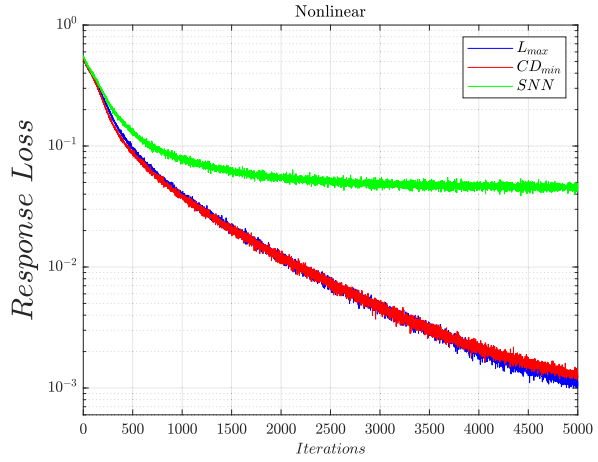

FIGURE 14. Response loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included.

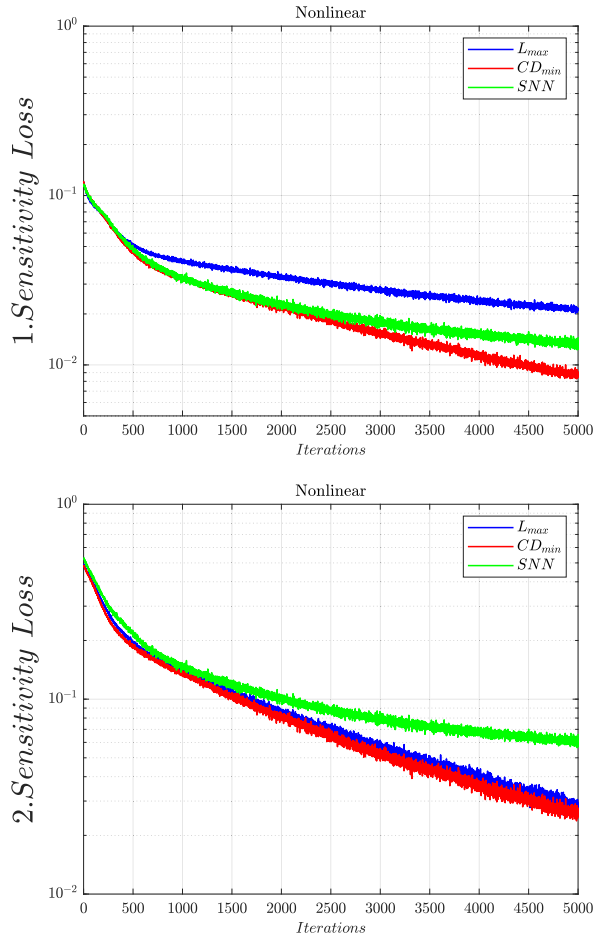

FIGURE 15. Top: First sensitivity loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included; Bottom: Second sensitivity loss over the training iterations of *max* $\mathcal{L}$ and *min* CD. For comparison the mode without residual weight optimization is included.

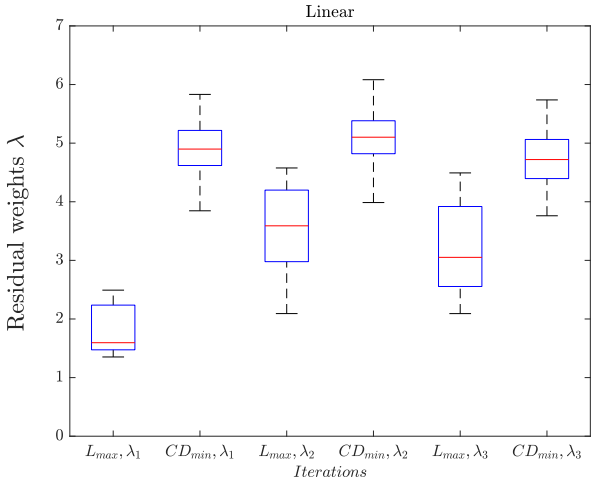

FIGURE 16. Spread of the final residual weight values of all training runs for *max* $\mathcal{L}$ and *min* CD.

of the spread of values in comparison to mode 2 and 10. However there is no strong performance improvement for the average, as well as the lowest obtained l_2 error. In Table. 5 the difference of performance is visualised with the average scalar l_2 error values. First, mode 6 is slightly outperforming the other modes, but not substantially. Surprisingly, mode 10, which is the basic Sobolev training mode, performs equally well, meaning unadjusted residual weights seem to perform just as well as the various applied methods. This is quite different compared to the linear results. The average training time of the modes behaves similar to the linear case. Mode 6 is the second to last longest training mode, with mode 2 and 4 showing similar ratios to it as in the linear case. In addition, mode 10 performing equally well, takes the lowest amount of training time. This very different result compared to the linear case is the reason for the additional analysis by expansion of epochs and layer structure. While the loss itself is no longer a good metric for comparison of modes, the modes 2 and 6 have a bigger loss than the basic Sobolev training, see Fig. 13. To actually judge the performance difference, Fig. 13 provides the l_2 error, where both modes outperform the basic Sobolev training mode, meaning again it takes less time to train a model for a certain accuracy with these modes. It can also be seen that with 1000 epochs the modes 2 and 6 outperform the basic Sobolev training mode. In addition, mode 6 outperforms mode 2. Compared to the 500 epoch training, there is again a performance difference, which will be discussed later in this section, in comparison to the 2000 epoch results. Interestingly, when observing the results in Fig. 14 and Fig. 15, mode 6 reduces the losses of all individual loss addends the best again, just like the linear case. Mode 2 seems to again have difficulties approximating the first sensitivity. Furthermore, the basic Sobolev training mode now seems to have difficulties keeping up performance of these three metrics. Still again, for the total loss, the residual weights are included in the calculation, which leads to the basic Sobolev training mode having the lowest loss of all three, as mentioned in Fig. 13. When analysing the values the residual weights end up with for the non-linear case, it can be seen in Fig. 16 that mode 2 again generally converges the residual weights to lower ranges than mode 6. In general, the

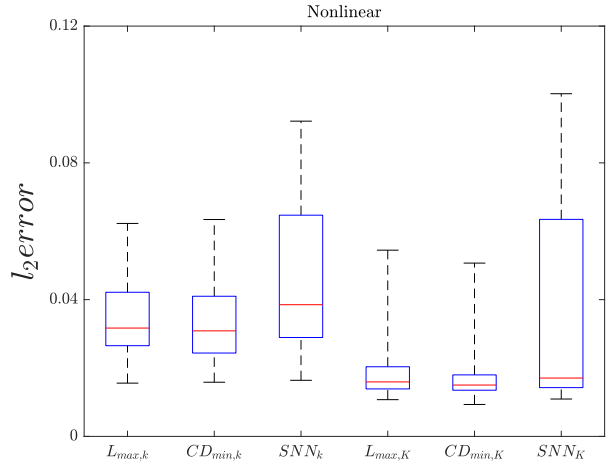

FIGURE 17. Spread of the l_2 error values of all training runs for layer structures $5-3-3$ and $10-5-5-3$ for *max* $\mathcal{L}$ and *min* CD.

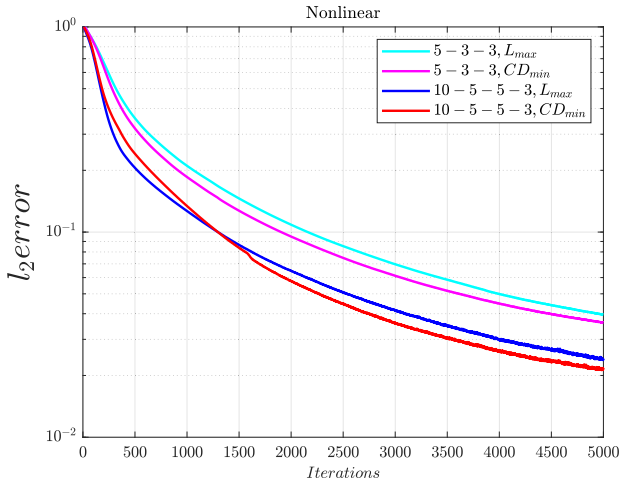

FIGURE 18. Curve of the l_2 error over the iterations for layer structures $5-3-3$ and $10-5-5-3$ for *max* $\mathcal{L}$ and *min* CD.

same behavior mentioned for the linear case is also found here for the non-linear case. The residual weights also stay in the same magnitude as the linear case. However the residual weights for the sensitivities for mode 2 seems to have a slightly bigger spread. Additionally, while mode 2 has the highest precision for the first residual weight, mode 6 seems to have similar behavior of precision for all three residual weights. It can be observed in Table. 6 that the final accuracy improves with epoch size. But the performance difference between *max* $\mathcal{L}$ and *min* $\boldsymbol{CD}$ does not seem to change with the epoch size. However, basic Sobolev training substantially increases its performance with the epoch size relative to the other two modes. Quite more interesting are the results for the expanded model layer structure. When considering Fig. 17, both mode 2 and 6 perform better. It can be noticed that the average l_2 error has improved better for the second layer structure "10-5-5-3" in both cases, however the difference between mode 2 and 6 seems to have expanded. Furthermore, the lowest obtained l_2 error has also improved for mode 6 in comparison to mode 2. The general superior performance of mode 6 compared to mode 2 over the training process can also be observed in Fig. 18. Generally, mode 6 produces better accuracy than mode 2 over the training iterations. Especially of note is Table. 7, where the performance difference between mode 2 and 6 can be seen to be expanding for a bigger layer structure. This shows, that the small model dimensions seem to bottleneck the performance of the gradient alignment method. Therefore, if the model were to be sufficiently sized, more than likely, the same behavior of results as the linear case should be obtained for the non-linear case.

VII. CONCLUSION

In this paper we applied the generated data from a FEM simulating the behavior of stress triaxiality response, as well as its sensitivity w.r.t. chosen geometric parameters, of a linear and non-linear material hook-shaped object under a defined set of force onto a neural network, making use of sensitivities. From here on various methods of finding the optimal set of residual weights applied to the individual losses of the gradient-enhanced neural network were observed. During comparison it was concluded that the method of minimising the cosine distance to align the gradient vectors of each individual loss with the total loss showed best convergence performance. Furthermore this gradient alignment method showed further improvement with expansion of layer width and depth. Specifically, the gradient alignment reduced the spread with an increased accuracy of the neural network training compared to Sobolev training. This was the case for both the linear and non-linear material data. Still, concerning the training time of the neural network, the gradient alignment method performed the worst. While the idea behind gradient alignment seems to work, there is great room for improvement in regards to optimization of the training algorithms computational efficiency. As a matter of fact, a more optimized and faster training variation has been tried and tested with similar accurate results. Another point of interest is the expansion of the layer structure, which could act as a bottleneck for the performance of the gradient alignment method. Finally, with respect to the applied method of minimising the cosine distance to optimise the residual weights $\boldsymbol{\lambda}$, it is understood that the residual weight adjusts the magnitude of the gradient vectors to affect the angle between the individual losses and the total loss. From this, introducing residual weights $\boldsymbol{\lambda}$ for each element of the gradient vectors would allow the algorithm to adjust not only magnitude but also the direction of the gradient vector. Thereby, the algorithm would ideally discard elements of gradient vectors which oppose each other and amplify elements that go hand in hand. However following that thought, the algorithm method would need to be reconsidered, as potentially minimising the cosine distance would have undesired effects of reducing convergence performance. Additionally, depending on the size of the neural network, having residual weights for each element results in more computational effort needed and should be considered carefully. In the early results, the method of aligning gradient vectors only to the response showed good results as well, which could be considered for

mixing with other methods. Another important goal in future research will be to apply the presented methods to expanded neural network models with, for example, convolutional architecture in order to capture the complete geometry of a given system and object. This would allow for more design variables and measured responses being evaluated in a single model, while also allowing for more complex, larger systems and neural network models to be tested with the proposed weighting methods.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the German Research Foundation (DFG) through the project "Damage Controlled Forming Processes" TRR 188-278868966.

REFERENCES

[1] G. I. Schuëller, "On the treatment of uncertainties in structural mechanics and analysis," *Comput. Struct.*, vol. 85, nos. 5–6, pp. 235–243, Mar. 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045794906003348
[2] M. Papadrakakis, V. Papadopoulos, and N. D. Lagaros, "Structural reliability analyis of elastic-plastic structures using neural networks and Monte Carlo simulation," *Comput. Methods Appl. Mech. Eng.*, vol. 136, nos. 1–2, pp. 145–163, Sep. 1996.
[3] J. E. Hurtado and D. A. Alvarez, "Neural-network-based reliability analysis: A comparative study," *Comput. Methods Appl. Mech. Eng.*, vol. 191, nos. 1–2, pp. 113–132, Nov. 2001.
[4] K. D. Kantarakias and G. Papadakis, "Sensitivity-enhanced generalized polynomial chaos for efficient uncertainty quantification," *J. Comput. Phys.*, vol. 491, Oct. 2023, Art. no. 112377.
[5] D. Bonnet-Eymard, A. Persoons, M. G. R. Faes, and D. Moens, "Quantifying uncertainty of physics-informed neural networks for continuum mechanics applications," in *Proc. 5th Int. Conf. Uncertainty Quantification Comput. Sci. Eng. (UNCECOMP)*, 2023, pp. 720–744.
[6] X. Feng and L. Zeng, "Gradient-enhanced deep neural network approximations," *J. Mach. Learn. Model. Comput.*, vol. 3, no. 4, pp. 73–91, 2022.
[7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.
[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 2, 2014, pp. 2672–2680.
[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989.
[13] S. K. Mitusch, S. W. Funke, and M. Kuchta, "Hybrid FEM-NN models: Combining artificial neural networks with the finite element method," *J. Comput. Phys.*, vol. 446, Dec. 2021, Art. no. 110651.
[14] T. Le-Duc, H. Nguyen-Xuan, and J. Lee, "A finite-element-informed neural network for parametric simulation in structural mechanics," *Finite Elements Anal. Des.*, vol. 217, May 2023, Art. no. 103904.
[15] R. E. Meethal, A. Kodakkal, M. Khalil, A. Ghantasala, B. Obst, K.-U. Bletzinger, and R. Wüchner, "Finite element method-enhanced neural network for forward and inverse problems," *Adv. Model. Simul. Eng. Sci.*, vol. 10, no. 1, p. 6, May 2023.
[16] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, vol. 25. Red Hook, NY, USA: Curran Associates, 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf
[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[18] L. Berrada, A. Zisserman, and M. P. Kumar, "Training neural networks for and by interpolation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 799–809.
[19] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data—A survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.
[20] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014.
[21] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations," 2017, *arXiv:1711.10561*.
[22] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019.
[23] L. Laurent, R. Le Riche, B. Soulier, and P.-A. Boucard, "An overview of gradient-enhanced metamodels with applications," *Arch. Comput. Methods Eng.*, vol. 26, no. 1, pp. 61–106, Jan. 2019.
[24] A. Bhaduri, D. Brandyberry, M. D. Shields, P. Geubelle, and L. Graham-Brady, "On the usefulness of gradient information in surrogate modeling: Application to uncertainty propagation in composite material models," *Probabilistic Eng. Mech.*, vol. 60, Apr. 2020, Art. no. 103024.
[25] M. G. Larson and F. Bengzon, *The Finite Element Method: Theory, Implementation, and Applications*, vol. 10. Springer, 2013, sec. 4.10, p. 97.
[26] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, "Sobolev training for neural networks," in *Advances in Neural Information Processing Systems*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Red Hook, NY, USA: Curran Associates, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/758a06618c69880a6cee5314ee42d52f-Paper.pdf
[27] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, "Regularisation of neural networks by enforcing Lipschitz continuity," 2018, *arXiv:1804.04368*.
[28] C. Finlay, J. Calder, B. Abbasi, and A. Oberman, "Lipschitz regularized deep neural networks generalize and are adversarially robust," 2018, *arXiv:1808.09540*.
[29] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, "Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems," *Comput. Methods Appl. Mech. Eng.*, vol. 393, Apr. 2022, Art. no. 114823. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045782522001438
[30] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Rev.*, vol. 63, no. 1, pp. 208–228, Jan. 2021.
[31] M. A. Bouhlel, S. He, and J. R. R. A. Martins, "Scalable gradient–enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes," *Struct. Multidisciplinary Optim.*, vol. 61, no. 4, pp. 1363–1376, Apr. 2020.
[32] T. G. Grossmann, U. Julia Komorowska, J. Latz, and C.-B. Schönlieb, "Can physics-informed neural networks beat the finite element method?" 2023, *arXiv:2302.04107*.
[33] F.-J. Barthold, N. Gerzen, W. Kijanski, and D. Materna, "Efficient variational design sensitivity analysis," in *Mathematical Modeling and Optimization of Complex Structures* (Computational Methods in Applied Sciences), vol. 40. Cham, Switzerland: Springer, 2016.
[34] J. Liedmann and F.-J. Barthold, "Variational sensitivity analysis of elastoplastic structures applied to optimal shape of specimens," *Struct. Multidisciplinary Optim.*, vol. 61, no. 6, pp. 2237–2251, Jun. 2020.
[35] A. O. M. Kilicsoy, "Gradient-enhanced neural networks: Applications in linear and nonlinear mechanics," M.S. thesis, Dept. Mech. Eng., CRE, Dortmund, Germany, 2023.
[36] L. McClenny and U. Braga-Neto, "Self-adaptive physics-informed neural networks using a soft attention mechanism," 2020, *arXiv:2009.04544*.
[37] D. Liu and Y. Wang, "A dual-dimer method for training physics-constrained neural networks with minimax architecture," *Neural Netw.*, vol. 136, pp. 112–125, Apr. 2021.
[38] K.-J. Bathe, *Finite—Elemente—Methoden*, 2nd ed., Berlin, Germany: Springer, 1990.
[39] P. Wriggers, *Nichtlineare Finite-Element-Methoden*. Springer-Verlag, 2013, sec. 5.2, p. 165.
[40] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method: Solid Mechanics*, vol. 2. London, U.K.: Butterworth, 2000.
[41] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method for Solid and Structural Mechanics*. Amsterdam, The Netherlands: Elsevier, 2005.

[42] T. J. R. Hughes, *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. New York, NY, USA: Dover, 2012.
[43] F.-J. Barthold, "Remarks on variational shape sensitivity analysis based on local coordinates," *Eng. Anal. Boundary Elements*, vol. 32, no. 11, pp. 971–985, Nov. 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0955799708000441
[44] J. Liedmann and F. Barthold, "Exploration of internal response sensitivities of materially nonlinear structures," *PAMM*, vol. 17, no. 1, pp. 745–746, Dec. 2017.
[45] J. Liedmann and F. J. Barthold, "Sensitivity analysis of elastoplastic structural response regarding geometry and external loads," in *Proc. 6th ECCM 7th ECFD*, 2018.
[46] J. T. Hoe, K. W. Ng, T. Zhang, C. S. Chan, Y.-Z. Song, and T. Xiang, "One loss for all: Deep hashing with a single cosine similarity based learning objective," in *Advances in Neural Information Processing Systems*, vol. 34, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., Red Hook, NY, USA: Curran Associates, 2021, pp. 24286–24298. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/cbcb58ac2e496207586df2854b17995fPaper.pdf
[47] H. V. Nguyen and L. Bai, "Cosine similarity metric learning for face verification," in *Computer Vision—ACCV 2010*, R. Kimmel, R. Klette, and A. Sugimoto, Eds., Berlin, Germany: Springer, 2011, pp. 709–720.
[48] B. Barz and J. Denzler, "Deep learning on small datasets without pre-training using cosine loss," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 1360–1369.

A. O. M. KILICSOY received the B.S. degree in mechanical engineering from Universität Duisburg-Essen, Duisburg, Germany, in 2020, and the M.S. degree in mechanical engineering from Technical University Dortmund (TU Dortmund), Dortmund, Germany, in 2023. Since 2023, he has been a Research Assistant with the Chair for Reliability Engineering, TU Dortmund. His research interests include neural networks, the design of simulation strategies for assessing component and system reliability in manufacturing processes, reliability sensitivity analysis, and machine learning.

J. LIEDMANN received the B.S. degree in civil engineering, the M.S. degree in structural engineering and the Ph.D. degree (summa cum laude) in computational mechanics from Technical University Dortmund (TU Dortmund), Dortmund, Germany, in 2011, 2014, and 2021, respectively. From 2014 to 2021, he was a Research Assistant with Lehrstuhl Baumechanik, TU Dortmund, where he has been a Postdoctoral Researcher, since 2021. His research interests include continuum mechanics, numerical methods and FEM (finite element method) formulations, structural optimization (topology and shape optimization), and variation formulations for structural and sensitivity analyses. He was awarded the Promotionspreis TU Dortmund, in 2022. He is a member of the International Society for Structural and Multidisciplinary Optimization (ISSMO), the International Association of Applied Mathematics and Mechanics (GAMM), the GAMM Student Chapter TU Dortmund, and European Scientific Association for Material Forming (ESAFORM).

M. A. VALDEBENITO received the Ph.D. degree in civil engineering from the University of Innsbruck, Innsbruck, Austria, in 2010. Since 2022, he has been the Chief Engineer at the Chair for Reliability Engineering, Technical University Dortmund, Dortmund, Germany. His research interests include the design of simulation strategies for assessing component and system reliability in structural mechanics, reliability sensitivity analysis, reliability-based design optimization, and uncertainty quantification under the presence of aleatoric and epistemic uncertainty. He is a member of the editorial board of the international journals *Computers and Structures* and *Structural Safety*. He was a recipient of the K. J. Bathe Award, in 2016, and a Humboldt Research Fellowship for Experienced Researchers.

F.-J. BARTHOLD received the Diploma degree in civil engineering from the University of Hannover, Germany, in 1986, M.Sc. degree in numerical analysis from Brunel University London, Uxbridge, U.K., in 1987, the Ph.D. degree in computational mechanics from the University of Hannover, in 1993, and the Habilitation degree from the Technical University of Braunschweig (TU Braunschweig), in 2001. From 1988 to 1999, he was a Research Assistant with the Institute for Structural Mechanics and Numerical Mechanics, University of Hannover. From 1999 to 2002, he was the Chief Engineer in computational sciences in engineering at TU Braunschweig and took on the role of an Interim Professor of engineering mechanics at the University of Kassel, from 2002 to 2003. Since 2003, he has been a Full Professor in computational mechanics at the Chair for Structural Mechanics in Civil Engineering (previously: Numerical Methods and Information Processing), Technical University Dortmund (TU Dortmund). His research interests include numerical methods in engineering, multiscale structural optimization and sensitivity analysis, continuum mechanics, and variational methods. In addition, he is an active member of several professional organizations, including the International Association of Applied Mathematics and Mechanics (GAMM), German Association for Computational Mechanics (GACM), the International Society for Structural and Multidisciplinary Optimization (ISSMO), European Mechanics Society (EUROMECH), and European Scientific Association for Material Forming (ESAFORM).

M. G. R. FAES received the B.S. degree (magna cum laude) in engineering technology from the Lessius University of Applied Sciences, Belgium, in 2013, the M.S. degree (summa cum laude) in engineering technology from the Thomas More University of Applied Sciences, St.-Katelijne-Waver, Belgium, and the Ph.D. degree in mechanical engineering from KU Leuven, Leuven, Belgium, in 2017. From 2013 to 2022, he was with the Department of Mechanical Engineering, KU Leuven. From 2013 to 2017, he was a Research Associate. From 2017 to 2018, he was Postdoctoral Fellow with the R2D Group. From 2018 to 2022, he was Postdoctoral Fellow of the Flemish Research Foundation working at the R2D Group. From 2020 to 2022, he was also Postdoctoral Fellow of the Alexander von Humboldt Foundation at the Institute for Risk and Reliability, Leibniz Universität Hannover, Niedersachsen, Germany. Since 2022, he has been a Professor in reliability engineering with the Chair for Reliability Engineering, Faculty of Mechanical Engineering, Technical University Dortmund (TU Dortmund), Dortmund, Germany. His research interests are inverse methods for uncertainty quantification, including interval techniques as well as Bayesian model updating schemes, advanced numerical propagation schemes for uncertainty analysis and quantification, reliability analysis and reliability-based design optimization under scarce data, and imprecise probabilistic concepts for robust uncertainty quantification. His awards and honors include the CIRP, JSEME Excellent Paper Award during the ISEM XVIII Conference on Electro Physical and Chemical Machining; the ECCOMAS Award for the two best Ph.D. theses on computational methods in applied sciences and engineering in Europe, in 2017; the SIPTA IJAR Young Researcher Award for research excellence in imprecise probabilities; the Willy Asselman Foundation Award for research excellence in honor of em. Prof. Willy Asselman; the EASD Junior Research Prize in the Area of Development of Methodologies for Structural Dynamics; and European Association of Structural Dynamics. He is an Associate Editor of the international journal *Mechanical System and Signal Processing*, as well as Associate Managing Editor of the *ASCE/ASME Journal for Risk and Uncertainty in Engineering Systems*. On top, he holds several more roles on editorial boards of international journals, as well as technical committee memberships at several international conferences.