Evaluation of Neural Network-based Derivatives for Topology Optimization

Joel Najmon
Student Member of ASME
School of Mechanical Engineering,
Purdue University,
585 Purdue Mall,
West Lafayette, IN 47907, USA
email: jnajmon@purdue.edu

Andres Tovar[1]
Professional Member of ASME
Associate Professor of Mechanical
Engineering, Purdue University in
Indianapolis
723 W Michigan St., SL 260N
Indianapolis, IN 46202, USA
email: tovara@iupui.edu

Neural networks have gained popularity for modeling complex non-linear relationships. Their computational efficiency has led to their growing adoption in optimization methods, including topology optimization. Recently, there have been several contributions towards improving derivatives of neural network outputs, which can improve their use in gradient-based optimization. However, a comparative study has yet to be conducted on the different derivative methods for the sensitivity of the input features on the neural network outputs. This paper aims to evaluate four derivative methods: analytical neural network's Jacobian, central finite difference method, complex step method, and automatic differentiation. These methods are implemented into density-based and homogenization-based topology optimization using multilayer perceptrons (MLPs). For density-based topology optimization, the MLP approximates Young's modulus for the solid-isotropic-material-with-penalization (SIMP) model. For homogenization-based topology optimization, the MLP approximates the homogenized stiffness tensor of a representative volume element, e.g., square cell microstructure with a rectangular hole. The comparative study is performed by solving two-dimensional topology optimization problems using the sensitivity coefficients from each derivative method. Evaluation includes initial sensitivity coefficients, convergence plots, and the final topologies, compliance, and design variables. The findings demonstrate that neural network-based sensitivity coefficients are sufficiently accurate for density-based and homogenization-based topology optimization. The neural network's Jacobian, complex step method, and automatic differentiation produced identical sensitivity coefficients to working precision. The study's open-source code is provided through a Python repository.

Keywords: Design Automation, Design Optimization, Machine Learning, Sensitivity Analysis for Design, Topology Optimization

1 Introduction

Over the past decades, neural networks have experienced a rapid surge in popularity as a means to model intricate non-linear relationships. They are particularly well-suited for this purpose because they can be trained solely using data without the need for an explicit function. Once trained, neural networks efficiently generate predictions, making them suitable for a wide range of applications. A typical application is for regression problems, in which the neural network is trained to predict the response of an expensive black-box function [1]. Moreover, the neural network can be employed in conjunction with various derivative methods to obtain approximate sensitivity coefficients of this function. Standard methods include analytical derivatives taken through the neural network's activation functions [2], the central finite difference method [3], the complex step method [4], and automatic differentiation [5].

The computational efficiency and flexibility of neural network-based derivatives have contributed to their increasing application to gradient-based optimization algorithms. This work specifically examines their use in structural topology optimization. In that context, neural network-based derivative methods play a valuable role in generating sensitivity coefficients enabling the efficient synthesis of the optimal material layout using gradient-based mathematical programming techniques [6].

1.1 Neural network-based derivatives. In recent years, a number of published studies have utilized and enhanced neural network-based derivatives and their associated numerical methods. Kissel and Diepold [7] proposed using least-squares approximated derivatives to train a Sobolev norm neural network for functions where target derivatives are not directly available. Meanwhile, Avrutskiy [8] showed that if several orders of target derivatives are known, then feedforward neural networks can be trained with increased precision. This was accomplished by incorporating deviations of the target derivatives from the known derivatives into an extended cost function. Later, Kiran and Naik [9] applied the complex step method to a feedforward neural network to obtain derivatives of a regression task accurately. Whereas Ledesma et al., [10] used the chain rule to take an analytical derivative of a multilayer perceptron neural network to derive a differential neural network. The differential neural network is based on the original network and does not need to be trained. Similarly, Rodini [11] derived and proposed a simple recursive algorithm for computing a deep neural network's first- and second-order derivatives via analytical methods.

Extending these methods to gradient-based optimization algorithms is straightforward, but its potential has not been fully explored or exploited. Notably, topology optimization is a compelling application for neural network-based derivatives, enabling the comparison of the sensitivity coefficients obtained using standard derivative methods and the corresponding final topologies.

1.2 Application to topology optimization. In recent years, neural networks have been applied to topology optimization in various ways, ranging from approximating the implicit function of a parameterized level set method [12] to generating three-dimensional aircraft models for design exploration [13]. A promising application is the prediction of effective properties of materi-

[1]Corresponding Author.
Version 1.18, November 28, 2023

als or microstructures, i.e., homogenized stiffness tensors, [14]. The resulting neural network material models can be applied to reduce the computational costs in density-based topology optimization (DBTO) and homogenization-based topology optimization (HBTO). As a result, these neural network material models enable the efficient evaluation of materials or microstructures that were previously computationally intractable in DBTO and HBTO methods.

DBTO methods use the density or thickness of each finite element as the design variable to optimize the material distribution. Several works have extended traditional DBTO methods with neural networks for predicting sensitivity coefficients or homogenized microstructures' properties. Takahashi et al. [15] proposed using convolution neural networks in topology optimization to predict the sensitivity coefficients from a discrete material distribution. Meanwhile, Watts et al. [16] deployed surrogate models to predict the homogenized stiffness tensor of open micro-trusses, given their relative densities. In a series of similar works, Zhang et al. [17–19] used the effective density of a microstructure to predict the homogenized properties of shape-interpolated microstructures generated with the parametric level set method.

HBTO methods use the parameters of a homogenized microstructure as the design variables to optimize the multiscale structure. HBTO methods date back to the seminal paper by [20] and commonly feature parameterizations including unit cells with rectangular holes [20–23], implicit functions [18,24,25], and truss structures [26–28]. These parameterizations facilitate extension (i.e., by providing input features) of HBTO methods with neural networks for predicting the homogenized microstructures' properties. White et al. [28] utilized single-layer feedforward neural networks to approximate the homogenized stiffness tensor of parametrically-sized micro-trusses. Similarly, Black et al. [29] presented a deep neural network to approximate the homogenized stiffness tensor of parametrically-shaped biotrusses. The homogenized properties of more complex microstructures have also been parameterized with implicit surface functions [30,31], for neural network approximation.

1.3 Objectives and contributions of this work. Although numerous derivative methods have been proposed and examined for neural networks, to the authors' knowledge, a comparative study to assess their performance has not been conducted. Therefore, the objective of this paper is to implement and evaluate standard methods for obtaining or approximating the derivatives of neural network outputs with respect to their inputs. The methods explored in this study encompass the analytical neural network Jacobian), the central finite difference method, the complex step method, and automatic differentiation. These methods are implemented and evaluated in neural networks trained to model material properties for DBTO and HBTO. Specifically, these neural networks are trained to predict the homogenized stiffness tensors of finite elements in a discretized design domain, given the design variables of the elements. The two topology optimization methods are used to solve the Messerschmitt-Bölkow-Blohm (MBB) beam problem using the sensitivity coefficients provided by each neural network-based derivative method. The results include the comparison of the initial sensitivity coefficients, convergence plots, final topologies, compliance values, and design variables. The training of the networks and evaluation of the derivative methods are done using the TensorFlow library in Python.

The DBTO method in this work utilizes a neural network material model trained to approximate the solid isotropic material with penalization (SIMP) model [32]. The SIMP model was chosen for several reasons. First, SIMP derivatives have a closed-form analytical expression, enabling a ground truth comparison. Second, it provides a single-input, single-output system that is well-suited to training an MLP. Third, the results are easier to verify and provide confidence in implementing the derivative methods. Finally, SIMP provides a familiar benchmark for the study. The HBTO method in this work utilizes a neural network material model trained to approximate the homogenized properties of a square cell microstructure with a rectangular hole. The microstructures are parameterized by the height and width of the hole, and their effective properties are found with numerical homogenization [33]. The HBTO method provides a multivariate input and output neural network application for a more comprehensive comparison of neural network-based derivative methods compared to the single input and output neural network for approximating the SIMP model.

The contributions of this work are the following: firstly, the implementation of four neural network-based derivative methods in TensorFlow that can be applied to a multilayer perceptron (MLP) of any arbitrary architecture. To the authors' knowledge, there is no other implementation of these methods made available to the scientific community via a general, open-source Python code. Secondly, the evaluation of four neural network-based derivative methods in terms of the relative accuracy of their sensitivity coefficients. Thirdly, this work is the first to assess the effect of neural network-based derivative methods on the final topologies generated by DBTO and HBTO. This study's DBTO and HBTO methods handle two-dimensional layouts of any used-defined size and boundary conditions. See Appendix A for a GitHub® repository of all the Python code utilized in this work.

The paper is organized as follows: the four studied derivative methods are presented in Sec. 2. The density-based and homogenization-based topology optimization methods and their neural network material models are detailed in Sec. 3. In Sec. 4, the neural network-based derivative methods are compared in the two topology optimization methods for the MBB beam problem. Lastly, the paper's conclusions are given in Sec. 5.

2 Neural Network-based Derivative Methods

Neural networks or artificial neural networks are a type of machine learning model that can be trained to recognize patterns or make predictions. Neural networks consist of interconnected neurons that perform weighted operations (i.e., additions and multiplications) passed through non-linear activation functions [34]. Each neuron's weight and bias variables can be optimized to minimize the loss or objective function of the model, e.g., the mean squared error between the target and the network's predicted output. Multiple layers of these neurons can be connected to form a deep neural network architecture. The simplest neural network architecture is found in a feedforward neural network where input data is only passed in the forward direction to the next layer. While many types of neural network architectures exist [35], this work employs a class of fully connected feedforward neural networks known as multilayer perceptrons (MLPs). This is motivated by the fundamental architecture of MLPs, which provides a traditional benchmark for the novel comparison of neural network-based derivative methods in this work.

Let us consider a simple MLP that consists of an input layer, an output layer, and N_L hidden layers (Fig. 1). The input and output layers have neurons that are equal to the input and output dimensions of the dataset to be trained, while each hidden layer has N_N neurons. The weight matrices $\mathbf{W}_n$ and bias vectors $\mathbf{B}_n$ of layer n modify the corresponding input $\mathbf{y}_{n-1}$ of the layer before the layer's activation function F_n is applied to calculate the output of the layer $\mathbf{y}_n$. This is given by

$$\mathbf{y}_n = F_n(\mathbf{W}_n \mathbf{y}_{n-1} + \mathbf{B}_n) \tag{1}$$

where n ranges from 1 to $N_L + 1$. For the input layer ($n = 0$), the input feature set $\mathbf{x}$ is equated to $\mathbf{y}_0$ and passed to the first hidden layer ($n = 1$). The output of the MLP is returned by the output layer ($n = N_L + 1$) as

$$\begin{aligned}\mathbf{y}_{N_L+1} = F_{N_L+1}\big(\mathbf{W}_{N_L} F_{N_L}\big(\ \cdots \\ \mathbf{W}_2 F_1\left(\mathbf{W}_1 \mathbf{x} + \mathbf{B}_1\right) + \mathbf{B}_2 \ \cdots \ + \mathbf{B}_{N_L}\big) + \mathbf{B}_{N_L}\big).\end{aligned} \tag{2}$$

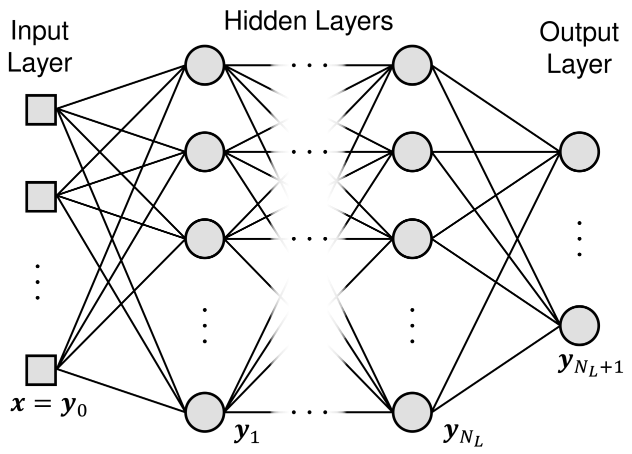


Fig. 1 Architecture of a simple multilayer perceptron feed-forward neural network. A circle indicates an element-wise computation of Eq. (1). A square indicates an input feature of the network.

While not portrayed in Fig. 1, distinct hidden layers may have differing N_N. Similarly, the activation function F_n can vary across the hidden and output layers. If normalization is applied to the features and targets, the first and last hidden layers can be defined with normalization and denormalization functions. In this case, the output of layers $\mathbf{y}_1$ and $\mathbf{y}_{N_L}$ will be given by these functions and not Eq. (1). The remainder of this section presents the four neural network-based derivative methods compared in this paper. See Appendix A for a general implementation of the four methods for an arbitrary MLP architecture.

2.1 Neural Network Jacobian. Since the output of a network is computed through a combination of linear operations passed through nested activation functions, an analytical derivative of the network's outputs can be computed with respect to its inputs by applying the chain rule. This is referred to as the neural network Jacobian (NNJ). Taking the derivative through the layers with respect to the input $\mathbf{x}$ yields

$$\frac{d\mathbf{y}_{N_L+1}}{d\mathbf{x}} = \frac{d\mathbf{y}_1}{d\mathbf{x}}\frac{d\mathbf{y}_2}{d\mathbf{y}_1}\cdots\frac{d\mathbf{y}_n}{d\mathbf{y}_{n-1}}\cdots\frac{d\mathbf{y}_{N_L}}{d\mathbf{y}_{N_L-1}}\frac{d\mathbf{y}_{N_L+1}}{d\mathbf{y}_{N_L}}. \tag{3}$$

The derivative of Eq. (1) with respect to the layer's input y_{n-1} is

$$\frac{d\mathbf{y}_n}{d\mathbf{y}_{n-1}} = \mathbf{W}_n \odot F_n'(\mathbf{W}_n\mathbf{y}_{n-1} + \mathbf{B}_n) \tag{4}$$

where F_n' is the derivative of the layer's activation function and $\odot$ is the Hadamard product with broadcasting support (i.e., NumPy's `multipy` function).

Substituting Eq. (4) into Eq. (3) we have

$$\frac{d\mathbf{y}_{N_L+1}}{d\mathbf{x}} = \mathbf{W}_1 \odot F_1'(\mathbf{W}_1\mathbf{x} + \mathbf{B}_1)\cdots\mathbf{W}_n \odot F_n'(\mathbf{W}_n\mathbf{y}_{n-1} + \mathbf{B}_n)$$
$$\cdots\mathbf{W}_{N_L+1} \odot F_{N_L+1}'(\mathbf{W}_{N_L+1}\mathbf{y}_{N_L} + \mathbf{B}_{N_L+1}). \tag{5}$$

Since $\mathbf{x} = \mathbf{y}_0$, Eqs. (3) and (5) can be expressed in Lagrange notation as

$$\frac{d\mathbf{y}_{N_L+1}}{d\mathbf{x}} = \prod_{n=1}^{N_L+1}\frac{d\mathbf{y}_n}{d\mathbf{y}_{n-1}} = \prod_{n=1}^{N_L+1}\mathbf{W}_n \odot F_n'(\mathbf{W}_n\mathbf{y}_{n-1} + \mathbf{B}_n). \tag{6}$$

See Appendix B for a detailed example of these calculations through a simple MLP.

2.2 Central Finite Difference Method. Due to the inexpensive function evaluations afforded by neural networks, the central finite difference method (CFD) can be efficiently applied to approximate derivatives. With CFD, the derivative of the network with respect to input x_j is approximated by

$$\frac{\partial\mathbf{y}_{N_L+1}}{\partial x_j} \approx \frac{\mathbf{y}_{N_L+1}\left(\mathbf{x} + h\boldsymbol{\delta}_j\right) - \mathbf{y}_{N_L+1}\left(\mathbf{x} - h\boldsymbol{\delta}_j\right)}{2h} \tag{7}$$

where h is a small real number (e.g., $h = 10^{-6}$) and $\boldsymbol{\delta}_j = [0,\ldots,0,1,0,\ldots,0]^\top$ with the number 1 located at the j^{th} row.

2.3 Complex Step Method. If the activation functions used in the MLP are holomorphic [36], then the complex step method (CSM) [37] may also be applied to approximate the derivatives. With CSM, the derivative of the network with respect to input x_j is approximated by

$$\frac{\partial\mathbf{y}_{N_L+1}}{\partial x_j} \approx \frac{\operatorname{Im}\left[\mathbf{y}_{N_L+1}\left(\mathbf{x} + \eta\boldsymbol{\delta}_j\right)\right]}{\eta} \tag{8}$$

where η is a small real number (e.g., $\eta = 10^{-12}$). Since the CSM does not involve a difference operation, it is not subjected to subtractive cancellation errors. This constitutes a significant advantage over finite difference-based approximations [4].

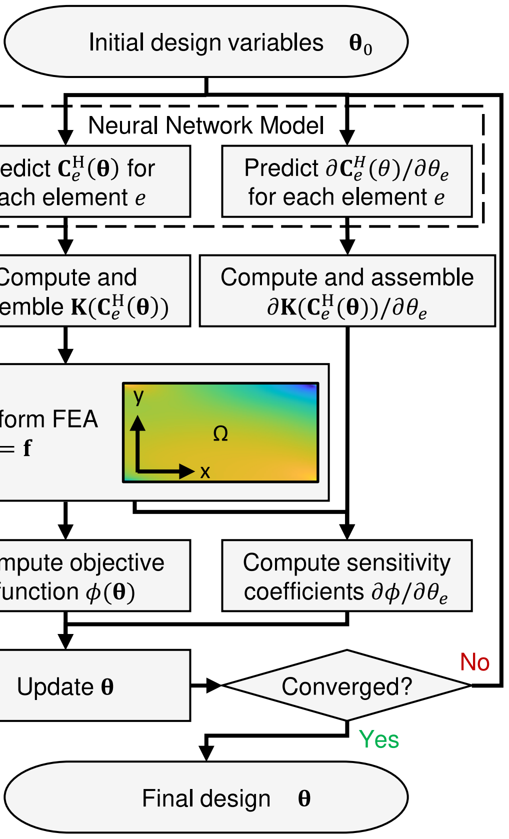
2.4 Automatic Differentiation. Automatic differentiation (AD) is a set of techniques to evaluate the derivative of a function defined by a computer program [38]. AD works by overloading standard elementary operators and functions with a derivative rule in addition to their function value. Similar to the previously derived NNJ, the chain rule can be repeatedly applied to these elementary operations, allowing for derivatives of arbitrary order to be computed automatically to the nominal working precision. The downside to AD is that it requires careful implementation into a software package [5], so its availability can be limited. TensorFlow natively supports AD [39], which makes its implementation straightforward.

3 Topology Optimization with Neural Networks

To evaluate the various neural network-based derivative methods of Sec. 2, neural networks are trained to approximate the material models in density-based and homogenization-based topology optimization. In the DBTO method, the neural network replaces the SIMP model, while in the HBTO method, the neural network replaces the model of an orthotropic square microstructure cell with a rectangular hole. Although these material models are relatively simple compared to those in other works [29], they provide a suitable context for studying neural network-based derivative methods; see Sec. 1.3 for motivation. To this end, this section presents an overview of the topology optimization method, the optimization problem statement, and the implementation of the neural network into the density-based and homogenization-based topology optimization methods.

3.1 Method Overview. In topology optimization methods with neural network material models, a neural network is first trained to predict the homogenized stiffness tensor $\mathbf{C}_e^{\mathrm{H}}$ of a finite element e given the design variables of that element. An initial design $\boldsymbol{\theta}_0$ is provided to start the optimization process. At every optimization iteration, $\mathbf{C}_e^{\mathrm{H}}$ is predicted for every element. After the global stiffness matrix $\mathbf{K}$ is assembled from the predicted stiffness tensors, finite element analysis (FEA) is performed to find the resulting displacements in the design domain Ω. The objective function can now be evaluated. Additionally, the derivative of the homogenized stiffness tensor with respect to each design variable $\partial\mathbf{C}_e^{\mathrm{H}}/\partial\theta_e$ is derived from the neural network with one of the four methods presented in Sec. 2. Following the same process for $\mathbf{K}$, the derivative of the stiffness matrix with respect to each design variable $\partial\mathbf{K}/\partial\theta_e$ is computed and assembled. The sensitivity coefficients can now be calculated using this and the FEA results.

The design variables $\boldsymbol{\theta}$ are then updated with the trust region method from SciPy's `optimize.minimize` function [40]. These steps occur at every optimization iteration until a final design (i.e., topology) has converged or a maximum number of iterations is reached. See Fig. 2 for a flowchart of the topology optimization method with a neural network material model.



Initial design variables $\boldsymbol{\theta}_0$
Neural Network Model
Predict $\mathbf{C}_e^{\mathrm{H}}(\boldsymbol{\theta})$ for each element e
Predict $\partial\mathbf{C}_e^{H}(\theta)/\partial\theta_e$ for each element e
Compute and assemble $\mathbf{K}(\mathbf{C}_e^{\mathrm{H}}(\boldsymbol{\theta}))$
Compute and assemble $\partial\mathbf{K}(\mathbf{C}_e^{\mathrm{H}}(\boldsymbol{\theta}))/\partial\theta_e$
Perform FEA $\mathbf{Ku} = \mathbf{f}$
y
x
Ω
Compute objective function $\phi(\boldsymbol{\theta})$
Compute sensitivity coefficients $\partial\phi/\partial\theta_e$
Update $\boldsymbol{\theta}$
Converged?
No
Yes
Final design $\boldsymbol{\theta}$

Fig. 2 Flowchart of the topology optimization method with a neural network material model. Inside the dashed window are subroutines that use the neural network material model.

3.2 Optimization Problem Statement. The topology optimization problem is to find the design variables $\boldsymbol{\theta}$ that minimize the compliance of the structure $\phi(\boldsymbol{\theta})$, while subject to a volume fraction constraint $v(\boldsymbol{\theta}) = V_t$ and satisfying the finite element model $\mathbf{Ku} = \mathbf{f}$ with the stiffness matrix $\mathbf{K}\left(\mathbf{C}_1^{\mathrm{H}}, \ldots, \mathbf{C}_e^{\mathrm{H}}, \ldots, \mathbf{C}_{n_e}^{\mathrm{H}}\right)$, where n_e is the total number of finite elements in the design domain Ω. The outer optimization problem statement is given by

$$\begin{aligned} \text{find} \quad & \boldsymbol{\theta} \in \theta_{\mathrm{ad}} \\ \min \quad & \phi(\boldsymbol{\theta}) = \mathbf{f}^{\top}\mathbf{u}(\boldsymbol{\theta}) \\ \text{s.t.} \quad & v(\boldsymbol{\theta}) = V_t, \end{aligned} \tag{9}$$

where θ_{ad} is a set of physically admissible variables, V_t is the target volume fraction, $\mathbf{f}$ is a vector of external forces on Ω and $\mathbf{u}(\boldsymbol{\theta})$ is a vector of displacements found by solving the finite element model.

Sensitivity coefficients are found indirectly via one of the four neural network-based derivative methods. First, the derivative of the network's output $\mathbf{C}_e^{\mathrm{H}}$ is found with respect to θ_e. The resulting $\partial\mathbf{C}_e^{\mathrm{H}}/\partial\theta_e$ terms are used to calculate

$$\frac{\partial\mathbf{K}}{\partial\theta_e} = \sum_{q=1}^{n_e}\int_{\Omega}\mathbf{B}_q^{\top}\frac{\partial\mathbf{C}_q^{\mathrm{H}}}{\partial\theta_e}\mathbf{B}_q\,dV, \tag{10}$$

where $\mathbf{B}_q$ is the element strain-displacement matrix and V is the volume of the design domain. Substituting Eq. (10) for $\mathbf{K}$ in the objective function of (9) we have the following sensitivity coefficients

$$\frac{\partial\phi(\boldsymbol{\theta})}{\partial\theta_e} = \mathbf{u}^{\top}\frac{\partial\mathbf{K}}{\partial\theta_e}\mathbf{u}. \tag{11}$$

The sensitivity coefficients are filtered with a neighborhood size of r to mitigate numerical problems such as checkerboard patterns and mesh dependencies [41].

For the DBTO method, the volume fraction constraint from Eq. (9) is defined as

$$v(\boldsymbol{\theta}) = \frac{1}{V}\sum_{e=1}^{n_e} v_e\theta_e = V_t, \tag{12}$$

where v_e is the volume of element e and the design variables $\boldsymbol{\theta}$ are the elemental densities. Conversely, for the HBTO method, the volume fraction constraint is defined as

$$v(\boldsymbol{\theta}) = 1 - \frac{\boldsymbol{\theta}_1{}^{\top}\boldsymbol{\theta}_2}{V} = V_t, \tag{13}$$

where $\boldsymbol{\theta}_1$ is the width and $\boldsymbol{\theta}_2$ is the height of the rectangular holes. The full set of design variables $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2]^{\top}$ are the parameters of this microstructure. This volume fraction constraint is nonlinear and, as such, had to be a supported feature of the applied optimization algorithm (i.e., trust-region method).

3.3 Neural Network Material Models. For the DBTO method, a single-input-single-output MLP is trained to surrogate the SIMP model's interpolation of Young's modulus

$$E(\theta_e) = E_{\min} + (E_{\max} - E_{\min})\theta_e^p, \tag{14}$$

given an input elemental density θ_e, where $E(\theta_e)$ is the interpolated Young's modulus for element e, p is the penalty parameter, and $E_{\min}$ and $E_{\max}$ are the minimum and maximum Young's modulus corresponding to a void and solid element, respectively. The predicted $E(\theta_e)$ value is utilized with a Poisson's ratio ν to define the Lamé constants for computing the isotropic stiffness tensor. The material properties are defined as $E_{\max} = 1$, $E_{\min} = 10^{-9}$, and $\nu = 0.3$. A penalization parameter of $p = 3$ is also defined. The set of physically admissible densities θ_{ad} is set to the range $[0, 1]$ for DBTO. Since the SIMP model, Eq. (14), is straightforward to differentiate, its analytical derivative

$$\frac{dE(\theta_e)}{d\theta_e} = -p(E_{\max} - E_{\min})\theta_e^{(p-1)}, \tag{15}$$

is utilized as the ground truth in the evaluation of the DBTO method (Sec. 4.2).

For the HBTO method, a two-dimensional input and four-dimensional output MLP is trained to surrogate the homogenized, orthotropic stiffness tensor of a microstructure with a rectangular hole (Fig. 3A) given the input width and height parameters $\boldsymbol{\theta}$. The homogenized stiffness tensor of the microstructure of macroscale element e is given by

$$\mathbf{C}_e^{\mathrm{H}} = \frac{1}{v_e}\sum_{f=1}^{n_f}\int_{v_e}\left(\mathbf{I} - \mathbf{B}_f\boldsymbol{\chi}_f\right)^{\top}\mathbf{C}_f\left(\mathbf{I} - \mathbf{B}_f\boldsymbol{\chi}_f\right)dv_e, \tag{16}$$

where $\boldsymbol{\chi}_f$ is the displacement fields from the unit strains shown in Fig. 3B, $\mathbf{I}$ is the identity matrix, $\mathbf{C}_f$ is the stiffness tensor of microscale element f, and n_f is the number of microscale elements [33]. The microstructures are discretized into 100×100 elements for numerical homogenization. The effect of this discretization can be seen in the contour plot of the homogenized stiffness tensor properties (Fig. 3C). The material properties defined during numerical homogenization are the same as those used in the DBTO method. Similarly, the set of physically admissible parameters θ_{ad} is set to the range $[0, 1]$ for HBTO.

4 Evaluation of Neural Network-based Derivative Methods in Topology Optimization

This section compares the four neural network-based derivative methods: NNJ, CFD, CSM, and AD. To this end, the MBB beam problem is solved using density-based and homogenization-based topology optimization. Additionally, the SIMP model, Eq. (14), and the analytical derivative of the SIMP model, Eq. (15), are utilized for a ground truth comparison when evaluating the DBTO method. Since Eq. (16) is approximated with numerical homogenization [33], its analytical derivative is computationally intractable. Therefore, a ground truth comparison is not available when evaluating the HBTO method. The results of this section compare the initial set of sensitivity coefficients for each method along with the final design variables, topologies, and compliance values.

4.1 Neural Network Training. Firstly, two neural networks, specifically MLPs, are trained to surrogate the material models of Sec. 3.3. To determine a suitable MLP architecture, the Keras-Tuner package was utilized to search the hyperparameters (e.g., N_L, N_N, learning rate, etc.) of the MLP using the hyperband algorithm [42]. After tuning, the optimal MLP architecture for both networks was found to be $N_L = 1$ hidden layers with $N_N = 64$ neurons in that layer. An optimal learning rate of 0.001 was found for the Adaptive Moment Estimation (Adam) optimization algorithm [43]. No kernel regularization was also found to produce better-performing MLPs. The hidden and output layers utilized the sigmoid and linear activation functions. The sigmoid activation function was selected over the relu activation function due to its increased smoothness. Normalization and denormalization layers are added before and after the hidden layers to improve training.

For each material model, the feature space was randomly sampled with lattice hypercube sampling to produce 10000 feature sets. This feature set size was motivated by the need to balance MLP accuracy and training time efficiently. The targets of these feature sets were found using Eq. (14) for the DBTO method's MLP or through numerical homogenization for the HBTO method's MLP. Each sampled dataset is broken into training (70%), testing (15%), and validation (15%) datasets, which are used for training with TensorFlow's `fit` function. The training process, which aims to minimize the mean squared error (MSE) between the targets and predictions of the MLP, spans for 5000 epochs with a batch size of 32. Note that no derivative information on the feature sets was provided for training the MLPs.

Training occurred on a workstation equipped with an 8-core Intel® Xeon® E5-2620 v4 @ 2.10GHz and 64 gigabytes of memory. Table 1 presents the training times t_t and testing performance metrics for the DBTO and HBTO MLPs with 10000 feature sets. An additional DBTO material model is trained with 100 feature sets to study the effect feature set density has on DBTO method's results. The DBTO model trained with 100 feature sets has the same feature set density (i.e., 100 feature sets per input dimension) as the HBTO model trained with 10000 feature sets. The MSE and correlation of determination (R^2) values show good agreement between the MLPs' prediction and the testing dataset, indicating no overfitting. As expected, a larger number of feature sets is shown to further improve the performance of the DBTO MLP at an additional computational cost. The performance metrics are observed to correspond more closely between the DBTO and HBTO MLPs if the feature set density is kept consistent. The MLPs that were trained with 10000 feature sets are always used in the following topology optimization problems unless stated otherwise. The GitHub® repository provided in Appendix A includes the datasets, MLP models, and training codes corresponding to this section.

Table 1 The training times and performance metrics for the DBTO and HBTO MLPs trained with 10000 feature sets and the DBTO MLP trained with 100 feature sets. The performance metrics include the mean squared error and correlation of determination between the MLPs' prediction and the testing dataset.

Metric	DBTO (10^2)	DBTO (10^4)	HBTO (10^4)
t_t	7.23 min	39.25 min	40.81 min
MSE	1.44×10^{-4}	1.87×10^{-7}	1.47×10^{-3}
R^2	0.99	1.00	0.99

4.2 Density-based Topology Optimization. The MBB beam's design domain Ω has a fixed support ($u_{1,2} = 0$) applied to the left side and a load ($f_2 = -1$) applied to the bottom corner of the right side (Fig. 4A). Ω is discretized into 60×30 elements. The sensitivity filter radius is set to 2.7 elements, and a volume fraction constraint of $V_t = 0.50$ is assigned. The MBB beam problem is solved with the DBTO method from an initial design (Fig. 4B) of $\boldsymbol{\theta} = V_t$ until the maximum change in $\boldsymbol{\theta}$ is less than 10^{-3}.

The final topologies for the MBB beam problem were solved with the DBTO method using the sensitivity coefficients from the four neural network-based derivative methods and the ground truth SIMP model (Fig. 5). The final compliance values and the number of iterations until convergence are also provided. For all of the derivative methods, the DBTO method converged to final topologies with a similar compliance value around $\phi = 82$. An additional MBB beam problem is solved using the DBTO method with the 100 feature set material model and AD sensitivity coefficients. Fig. 6 compares the result of this problem with the result from Fig. 5 using AD sensitivity coefficients. The final topologies are similar, but the more accurate 10000 feature set material model (Table 1) produced a final topology with better compliance that is closer to the ground truth SIMP. The DBTO method's convergence for most derivative methods was also very similar to the sensitivity coefficients provided by the ground truth SIMP model (Fig. 7). However, the sensitivity coefficients from the CSM method resulted in a significantly different and more slowly converging optimization problem.

To compare the very similar but still unique final topologies in Fig. 5, a matrix of topology difference plots is provided in Fig. 8. Each plot corresponds to the difference between the final topologies of the column's method and the row's method. The MSE between the densities $\boldsymbol{\theta}_{\text{MSE}}$ of each topology comparison is also provided. The smallest difference between the final topologies is seen between the NNJ, CFD, and AD methods. Due to the significantly different path to convergence, the CSM method had the most unique final topology. Compared to using the SIMP model, the NNJ, CFD, and AD methods all had a similar error.

All of the comparisons made so far have dealt with differences in the final topologies. The differences between these final topologies are amplified by initial differences in the neural network-based sensitivity coefficients that get compounded and grown over the numerous optimization iterations. Therefore, while the comparisons between the final results of the DBTO method provide useful insight into how the derivative method influences the convergence and final topology, it does not provide insight into how the neural

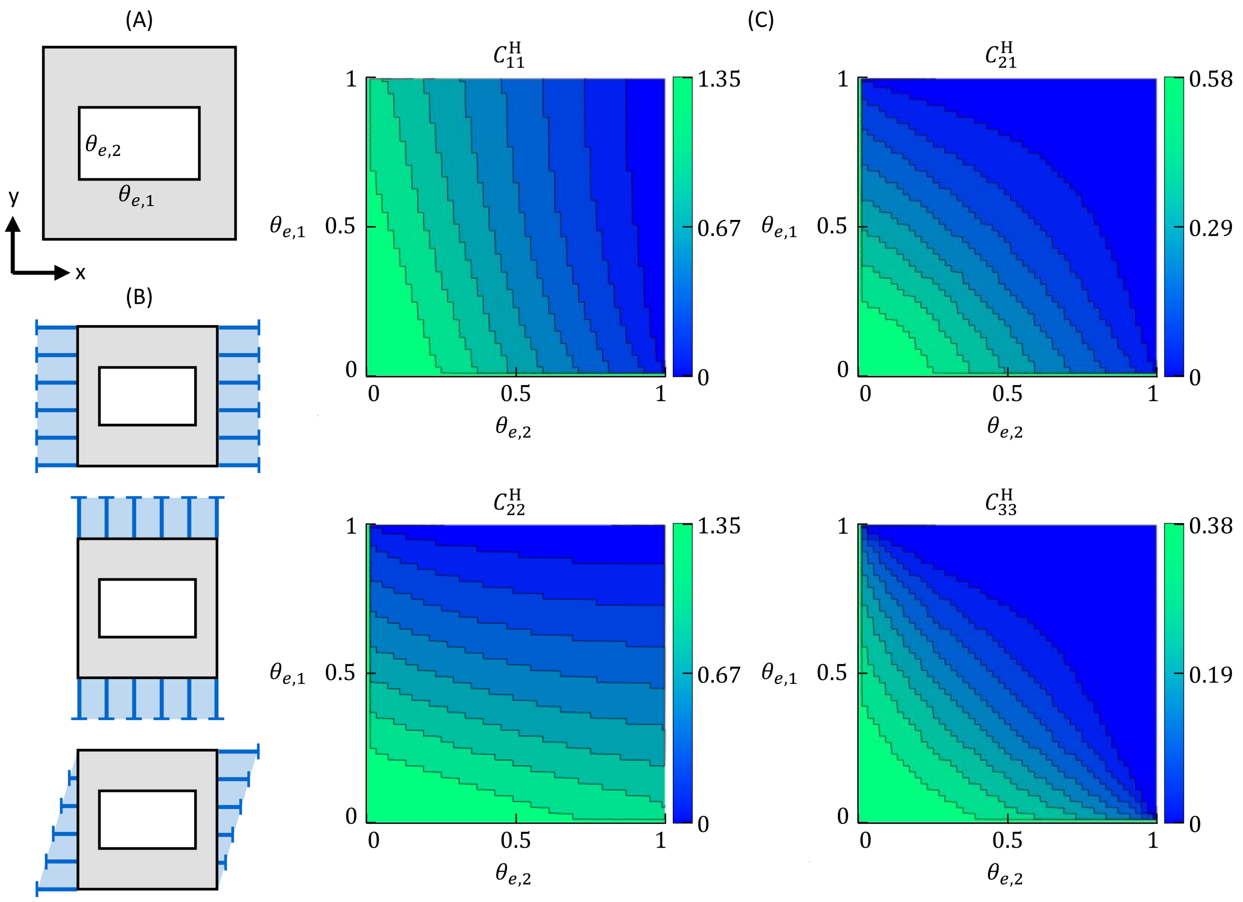


Fig. 3 The (A) microstructure with a rectangular hole, (B) loading cases for numerical homogenization, and (C) contour plots of the homogenized, orthotropic stiffness tensor properties.

network-based derivative methods actually differ from one another on a functional level.

For this reason, the MSE between the sensitivity coefficients of the initial design $\partial\phi_0/\partial\boldsymbol{\theta}_{\text{MSE}}$ of each topology comparison are also provided in Fig. 8. An interesting observation is that there is zero error between the NNJ, CSM, and AD sensitivity coefficients, at least near the nominal working precision of the computer. In theory, the convergence and results of the DBTO method with these coefficients should be identical; however, this is not the case due to an accumulation of numerical (truncation and round-off) errors over the optimization iterations. These three sets of equal sensitivity coefficients agree with the true sensitivity coefficients from the SIMP model. The CFD method is the only method that produces sensitivity coefficients different from the other derivative methods, most likely due to the approximation that occurs from its small finite step ($h = 10^{-6}$) in Eq. (7). With respect to the sensitivity coefficients and final topologies from the DBTO method, the most consistent neural network-based derivative methods are NNJ and AD.

4.3 Homogenization-based Topology Optimization. The same boundary conditions, optimization settings, and convergence criteria are applied to the HBTO method as what was used in the DBTO method. The MBB beam problem is solved accordingly using an initial design (Fig. 4C) that assumes equal parameters $\boldsymbol{\theta}$ across the design domain Ω, which satisfy the volume fraction constraint $V_t = 0.50$.

The multiscale topologies for the MBB beam at 10 iterations, 100 iterations, and the final iteration (i.e., a little over 1000 iterations) are shown in Fig. 9 for the HBTO method using the sensitivity coefficients from the four neural network-based derivative methods. The compliance values for each topology are also provided. The multiscale topologies are assembled from the microstructures with rectangular holes specified by $\boldsymbol{\theta}$. For plotting, each microstructure is discretized by a 100×100 element mesh. The compliance values of the topologies at 10 iterations were equal for the NNJ, CSM, and AD methods. As before, the compliance of the final topologies for all four methods converged to a similar compliance value around $\phi = 76$. The convergence of the HBTO method was similar for all four methods, with the greatest similarity occurring between the NNJ and AD methods (see Fig. 10). Despite the different number of iterations for convergence between these two methods, they converged to a topology with the same compliance value $\phi = 75.70$. The CFD method had the most different compliance values and convergence plots compared to the other three methods, which were all similar.

As with the DBTO results, a matrix of final topology difference plots is provided in Fig. 11. The topology difference plot is computed between the assembled multiscale topologies, not the final parameters $\boldsymbol{\theta}$ used to create the topologies. However, an MSE is computed between the parameters $\boldsymbol{\theta}_{\text{MSE}}$ for each topology comparison. The smallest difference between the final topologies is seen between the NNJ, CSM, and AD methods. As seen in the DBTO method, the derivative method with the most different convergence plots had the most different final topology (i.e., the CFD method).

As before, the MSE between the sensitivity coefficients of the initial design $\partial\phi_0/\partial\boldsymbol{\theta}_{\text{MSE}}$ of each topology comparison are provided in Fig. 11. Here we observe the same phenomenon seen in the DBTO method, namely, that the sensitivity coefficients resulting from NNJ, CSM, and AD are equal, i.e., $\partial\phi_0/\partial\boldsymbol{\theta}_{\text{MSE}} = 0$. As previously mentioned, the convergence and final results of the HBTO method are different due to an accumulation of numerical

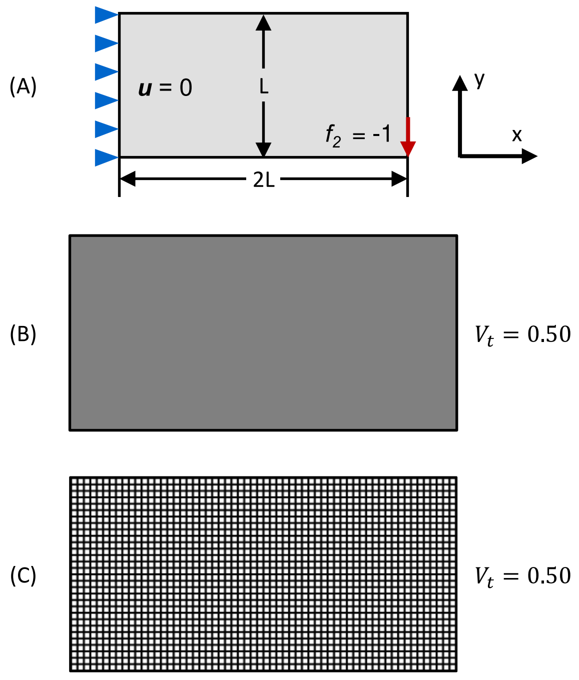


Fig. 4 The (A) boundary conditions of the MBB beam and its initial designs for the (B) DBTO and (C) HBTO methods. The DBTO method's initial design is defined by $\theta = V_t$. The HBTO method's initial design is defined by $\theta = \sqrt{1 - V_t}$ so that the microstructure of Fig. 3A has a square hole sized according to V_t.

errors over the optimization iterations. The effect of this error accumulation can be seen in Fig. 9, where the compliance values for the NNJ, CSM, and AD methods are no longer equal after 100 iterations. The two most consistent neural network-based derivative methods are the NNJ and AD methods when considering the final compliance value, convergence plots, initial sensitivity coefficients, and similarity in final design variables. For further reference, the average computational time it takes to evaluate the sensitivity coefficients per optimization iteration is reported in Table 2 for the four neural network-based derivative methods. On average, the fastest derivative method is CSM, while the slowest derivative methods were NNJ and AD for the DBTO and HBTO methods, respectively. That being said, the implementations of the four derivative methods (see Appendix A) have not been optimized for efficiency, and therefore, the reported times are not definitive on which method is the fastest.

Table 2 The average time spent computing sensitivity coefficients at each optimization iteration for the four neural network-based derivative methods. The computational cost is greater for the HBTO method due to having double the number of design variables. The times are measured on the same workstation specified in Sec. 4.1.

Derivative Method	DBTO	HBTO
NNJ	2.741 s	2.988 s
CFD	0.342 s	0.682 s
CSM	0.088 s	0.180 s
AD	2.568 s	8.553 s

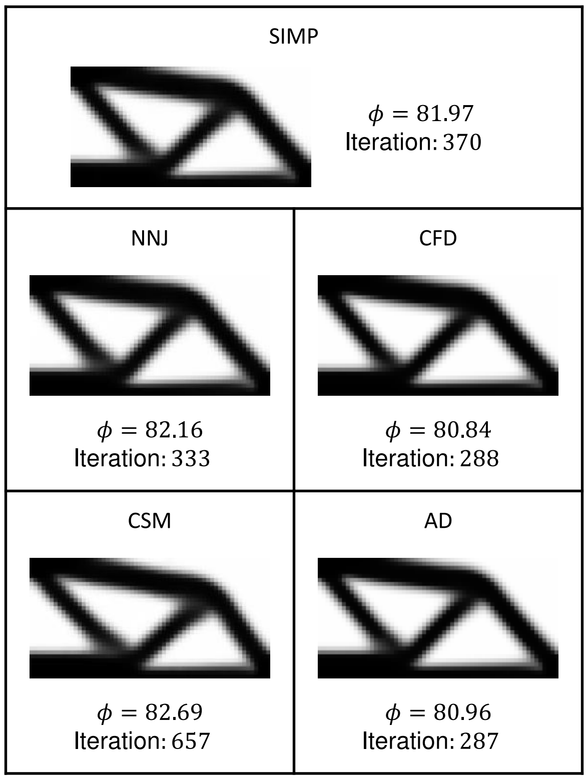


Fig. 5 The final topologies for the MBB beam problem were solved with the DBTO method using the sensitivity coefficients from the four neural network-based derivative methods and the ground truth SIMP model. The final compliance and iterations till convergence are also shown.

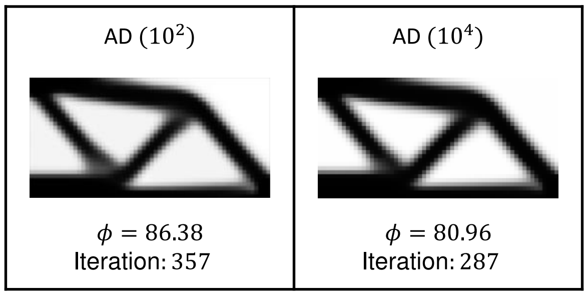


Fig. 6 The final topologies for the MBB beam problem were solved with the DBTO method using the material models trained with the 100 and 10000 feature sets. Sensitivity coefficients are approximated with AD for both cases. The final compliance and iterations till convergence are also shown.

5 Conclusion

This paper implements and compares four neural network-based derivative methods utilized to produce the sensitivity coefficients for two traditional topology optimization methods. The derivative methods studied include NNJ, CFD, CSM, and AD. The topology optimization applications include DBTO and HBTO. For each topology optimization method, a neural network is trained to approximate the material model of the method. For DBTO, a one-dimensional input and one-dimensional output MLP are trained to approximate the SIMP model's interpolation of Young's modulus. For HBTO, a two-dimensional input and four-dimensional output MLP is trained to approximate the homogenized and orthotropic stiffness tensors of microstructures with rectangular holes. These

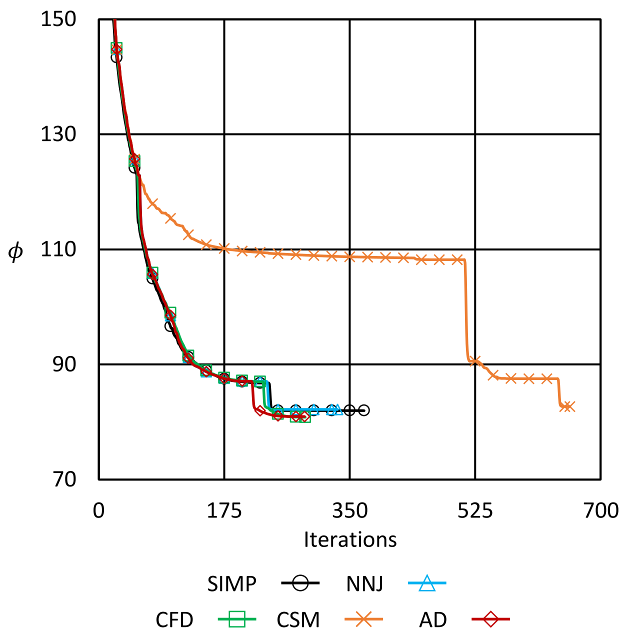


Fig. 7 Convergence plots of the MBB beam's compliance for the DBTO method when using sensitivity coefficients from the different neural network-based derivative methods and the ground truth SIMP model. See Fig. 5 for the corresponding final topologies.

neural network material models are implemented in the topology optimization methods to predict the material properties and their corresponding sensitivity coefficients at each iteration. A general implementation of the four neural network-based derivative methods for an arbitrary MLP architecture has also been included for further reference.

The single-layer MLPs for both material models were tuned, trained, and tested using only sampled data from the models, i.e., no derivative information was provided. The resulting networks agreed well with the predicted material model and the testing dataset. After training, the comparative study is carried out by solving the MBB beam problem with the DBTO and HBTO methods using sensitivity coefficients supplied by each derivative method. Comparisons are made between the initial sensitivity coefficients, convergence plots, final topologies, compliance values, and design variables.

Several key observations were made from the comparative studies of both topology optimization applications. Firstly, all four of the derivative methods produced final topologies and design variables that were similar to one another (e.g., MSE $\leq 7.58 \times 10^{-2}$). Secondly, the derivative methods that produced the most different final topologies and design variables had the most different convergence paths. These derivative methods are CSM and CFD for the DBTO and HBTO, respectively. Thirdly and most surprisingly, the NNJ, CSM, and AD methods produced the exact same sensitivity coefficients for the initial design of the MBB beam, at least near the nominal working precision of the computer. The CFD method produced similar sensitivity coefficients with a small MSE error to the other methods. Lastly, the NNJ and AD methods were the most similar methods regarding their convergence paths, final topologies, compliance values, and sensitivity coefficients. As such, these two derivative methods are recommended for optimization applications.

In the context of topology optimization methods with neural network material models, the NNJ, CSM, and AD methods are all capable of approximating the problem's sensitivity coefficients with a well-trained MLP, regardless of the number of input and output dimensions. These findings differ slightly from our preliminary results [44], which indicated that all of the derivative methods were capable of approximating a network's derivative to near working precision of the computer; however, this study was done in the context of networks trained to predict one-dimensional analytical functions with a single variable input. Additionally, a more accurate MLP material model was shown in the DBTO method to produce final topologies that were better performing and closer to the ground truth. Lastly, this work demonstrates that neural network-based sensitivity coefficients are sufficient for density-based and homogenization-based topology optimization methods.

From the results of this study, several recommendations can be made regarding neural network-based derivative methods. In most cases, NNJ and AD are the primary derivative methods recommended as they had the lowest error in sensitivity coefficients and showed the closest agreement to one another, even over numerous optimization problems. More specifically, if a software package natively supports AD, it is recommended as its utilization is significantly simpler than implementing NNJ. On the other hand, if AD is not supported, NNJ is recommended as it is far easier for an end-user to implement it into an existing software package or an in-house code. The other derivative methods (CFD, CSM) are valuable when computing the analytical derivative through the network involves discontinuous activation functions or when access to the network's hyperparameters and other details is restricted, e.g., when interfacing with third-party networks. In particular, in the case of holomorphic activation functions, CSM will be preferred over CFD.

The trust region algorithm utilized in this work is slower to converge in the topology optimization problems studied. Consequently, a large number of iterations are required to solve each problem. This is most noticeable in the HBTO method, where certain parameters (e.g., rectangular hole height) near the boundary result in poorly connected microstructure. Additionally, since a ground truth comparison was unavailable for the HBTO method, the conclusions derived from Sec. 4.3 are based only on the numerical approximations provided by the four derivative methods. Lastly, this work employs dense MLPs as the neural network for the comparative study. Therefore, any conclusions from this paper regarding neural network-based derivatives should be taken with this architecture in mind.

Acknowledgment

This manuscript is based upon work supported by the National Science Foundation, Candent Technologies, and the U.S. Naval Air Systems Command. The authors gratefully acknowledge their support. Any opinions, findings, conclusions, or recommendations expressed in this manuscript are those of the authors and do not necessarily reflect the views of our supporters.

Funding Data

- National Science Foundation (Grant No. 1842164)
- U.S. Naval Air Systems Command (STTR Contract No. N68335-22-C-0448)

Appendix A: Python Codes

This work was primarily developed and deployed in Python. To aid the reader in reproducing our results, the source code is provided in the following GitHub® repository: github.com/edrl-purdue/nn-derivative-to. The repository includes the MLP models utilized in this work and their training scripts, the DBTO and HBTO methods, and the implementation of the four neural network-based derivative methods for an arbitrary MLP architecture. The specific files and their function are:

- `NN_derivatives_examples.py`: This script provides a general implementation of the four derivative methods of Sec. 2 with several multivariate examples.

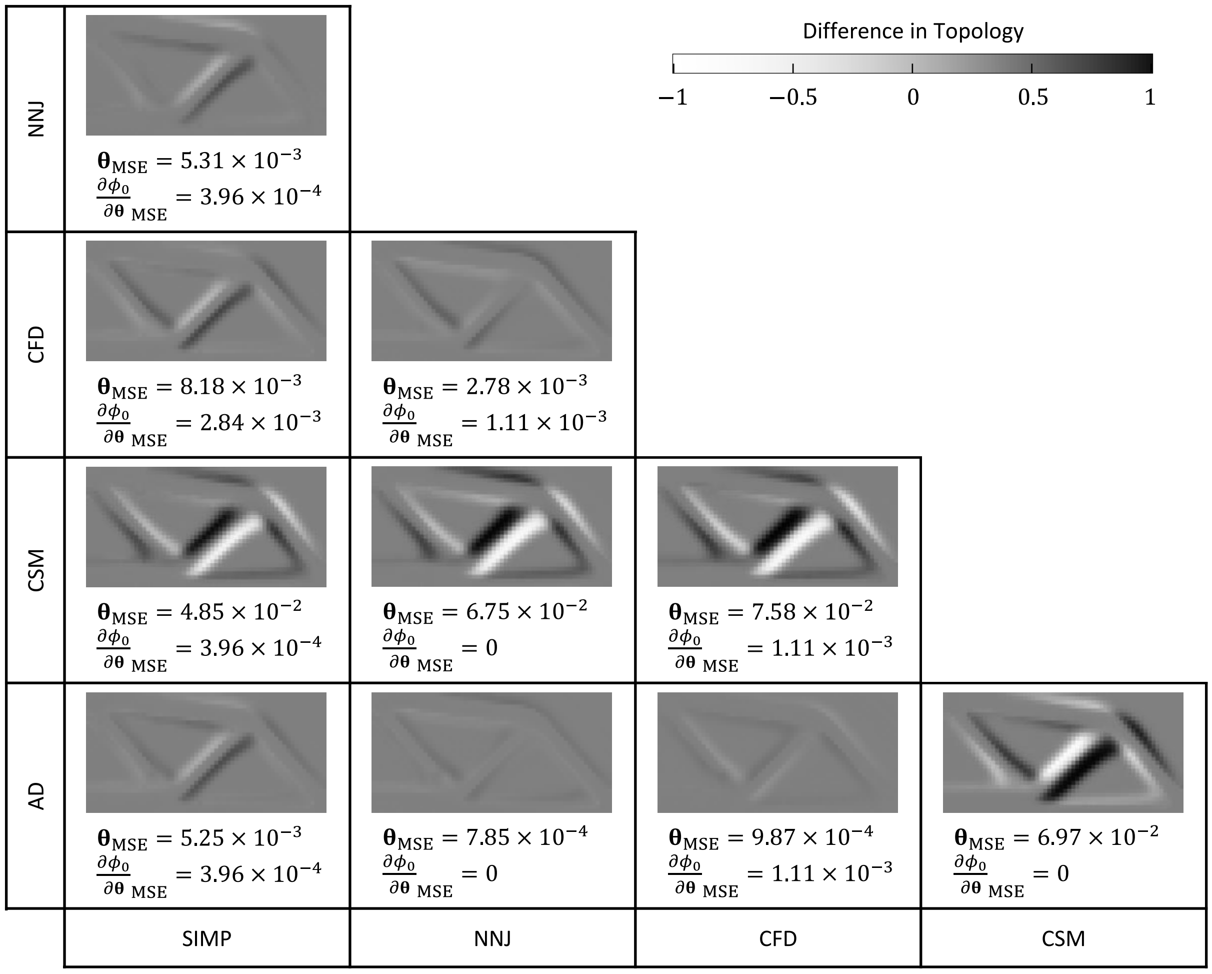


Fig. 8 The matrix of topology difference plots between all of the final topologies of the DBTO-solved MBB beam problem (Fig. 5). Each topology difference plot corresponds to the difference between the final topology of the column's method (bottom row) and the final topology of the row's method (left column). The MSE between the densities $\boldsymbol{\theta}_{\text{MSE}}$ of each comparison (i.e., plot) is provided. Additionally, the MSE between the sensitivity coefficients of the initial design $\partial\phi_0/\partial\boldsymbol{\theta}_{\text{MSE}}$ of each comparison is also provided.

- `Train_DBTO_NN.py`: This script trains the neural network material model for DBTO.
- `Train_HBTO_NN.py`: This script trains the neural network material model for HBTO.
- `Run_DBTO_NN.py`: This script executes DBTO on the MBB beam example using the neural network material model.
- `Run_DBTO_SIMP.py`: This script executes DBTO on the MBB beam example using the SIMP material model.
- `Run_HBTO_NN.py`: This script executes HBTO on the MBB beam example using the neural network material model.
- `Appendix_B_example.py`: This script trains the MLP and performs the NNJ calculations in Appendix B.

Appendix B: Example of Neural Network Jacobian Calculations

This appendix presents an example of the calculations for the Jacobian (Sec. 2.1) of a neural network's output with respect to its input. Let us consider the following MLP that has been trained to approximate the target function

$$y = x^2, \tag{B1}$$

where x is the input variable ranging between $[-1, 1]$. The MLP has two hidden layers $N_L = 2$ with three neurons per hidden layer $N_N = 3$ as shown in Fig. 12. The MLP is well trained with a performance MSE of 1.45×10^{-6}; see Appendix A for a deterministic Python script to train this MLP.

From Eq. (6) the NNJ of the MLP can be calculated with

$$\frac{d\mathbf{y}_3}{d\mathbf{x}} = \frac{d\mathbf{y}_1}{d\mathbf{y}_0}\frac{d\mathbf{y}_2}{d\mathbf{y}_1}\frac{d\mathbf{y}_3}{d\mathbf{y}_2}, \tag{B2}$$

where

$$\frac{d\mathbf{y}_1}{d\mathbf{y}_0} = \mathbf{W}_1 \odot F_1'(\mathbf{y}_0\mathbf{W}_1 + \mathbf{B}_1), \tag{B3}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \mathbf{W}_2 \odot F_2'(\mathbf{y}_1\mathbf{W}_2 + \mathbf{B}_2), \tag{B4}$$

and

$$\frac{d\mathbf{y}_3}{d\mathbf{y}_2} = \mathbf{W}_3 \odot F_3'(\mathbf{y}_2\mathbf{W}_3 + \mathbf{B}_3) \tag{B5}$$

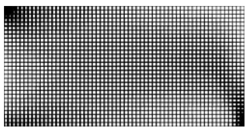
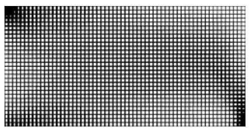
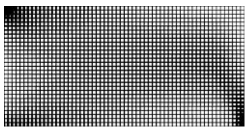
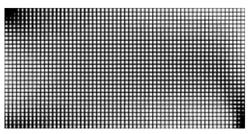
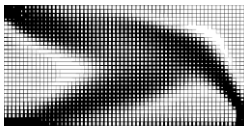
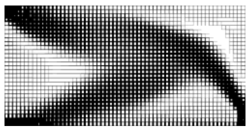
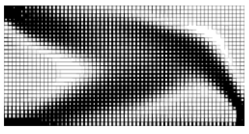
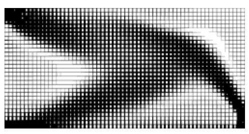
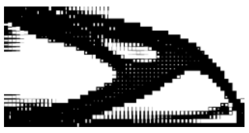
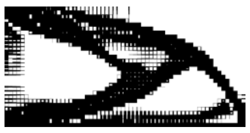
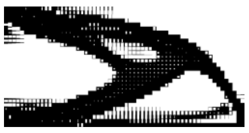
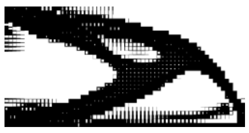
	NNJ	CFD	CSM	AD
Iteration: 10	 ϕ = 172.68	 ϕ = 173.83	 ϕ = 172.68	 ϕ = 172.68
Iteration: 100	 ϕ = 105.19	 ϕ = 106.98	 ϕ = 105.23	 ϕ = 105.74
Final	 ϕ = 75.70 Iteration: 1225	 ϕ = 76.78 Iteration: 1105	 ϕ = 76.14 Iteration: 1371	 ϕ = 75.70 Iteration: 1342

Fig. 9 The topologies at 10, 100, and the final iteration for the MBB beam problem solved with the HBTO method using the sensitivity coefficients from the four neural network-based derivative methods. The final compliance and iterations till convergence are also shown. θ within 7.5% of their bounds are set to the bounds to avoid very small holes or thin members.

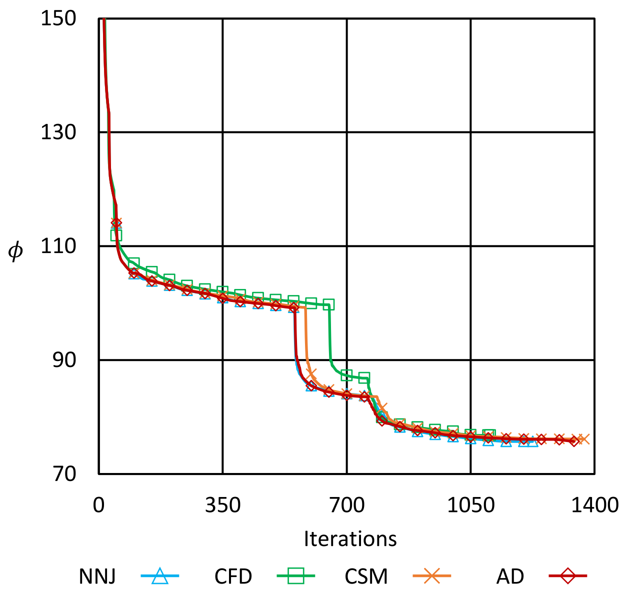


Fig. 10 Convergence plots of the MBB beam's compliance for the HBTO method when using sensitivity coefficients from the different neural network-based derivative methods. See Fig. 9 for the corresponding final topologies.

are derivatives of each layer with respect to its input, Eq. (4).

Let us first determine the derivatives of the activation functions F_1', F_2', and F_3'. The two hidden layers ($n = 1, 2$) of the MLP use the sigmoid activation function

$$F_1(t) = F_2(t) = \frac{1}{1 + e^{-t}}, \tag{B6}$$

which has the following derivative with respect to its input t

$$F_1'(t) = F_2'(t) = \frac{e^{-t}}{(1 + e^{-t})^2}. \tag{B7}$$

Conversely, the output layer ($n = 3$) uses the linear activation function

$$F_3(t) = t. \tag{B8}$$

and has a constant derivative

$$F_3'(t) = 1. \tag{B9}$$

Substituting Eq. (1) (i.e., the output of the hidden layers), Eq. (B9), and $\mathbf{y}_0 = \mathbf{x}$ into Eqs. (B3), (B4), and (B5) we have

$$\frac{d\mathbf{y}_1}{d\mathbf{y}_0} = \mathbf{W}_1 \odot F_1'(\mathbf{x}\mathbf{W}_1 + \mathbf{B}_1), \tag{B10}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \mathbf{W}_2 \odot F_2'((F_1(\mathbf{x}\mathbf{W}_1 + \mathbf{B}_1))\mathbf{W}_2 + \mathbf{B}_2), \tag{B11}$$

and

$$\frac{d\mathbf{y}_3}{d\mathbf{y}_2} = \mathbf{W}_3. \tag{B12}$$

The next step is to substitute in the weight matrices and bias vectors from the trained MLP. Accordingly, we have the following weight matrices

$$\mathbf{W}_1 = \begin{bmatrix} 1.735 & -0.850 & 1.404 \end{bmatrix}, \tag{B13a}$$

$$\mathbf{W}_2 = \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix}, \tag{B13b}$$

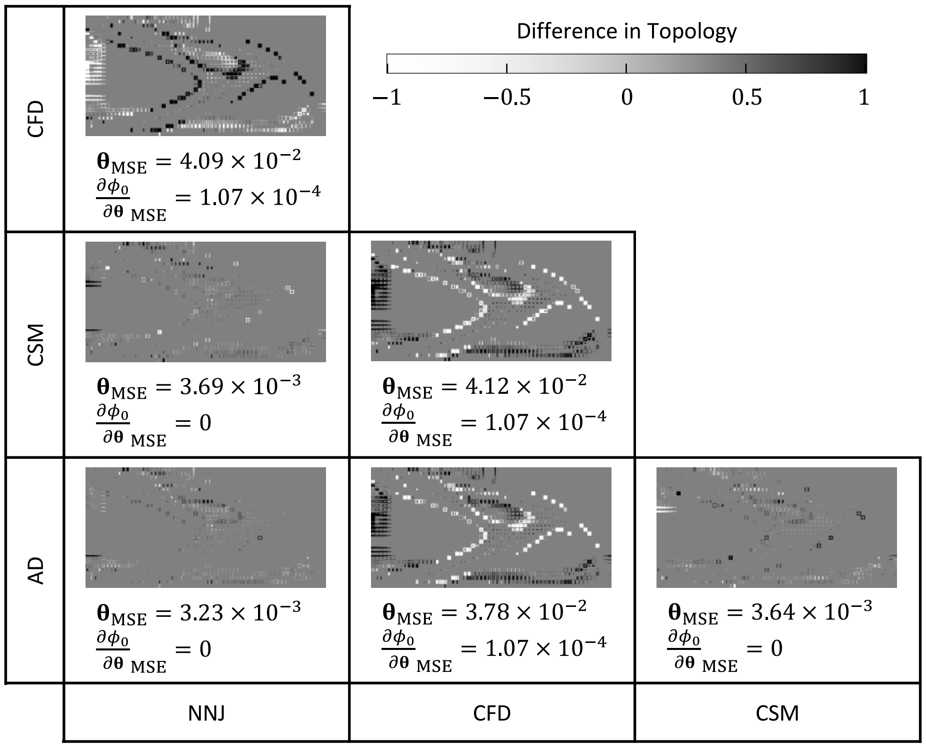


Fig. 11 The matrix of topology difference plots between all of the final topologies of the HBTO-solved MBB beam problem (Fig. 9). Each topology difference plot corresponds to the difference between the final topology of the column's method (bottom row) and the final topology of the row's method (left column). The MSE between the parameters θ_{MSE} of each comparison is provided. Additionally, the MSE between the sensitivity coefficients of the initial design $\partial\phi_0/\partial\theta_{\mathrm{MSE}}$ of each comparison is also provided.

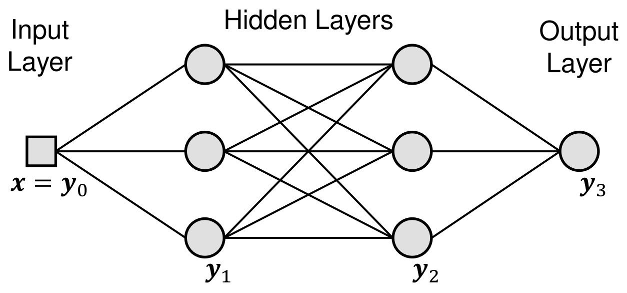


Fig. 12 Architecture of the multilayer perceptron feedforward neural network used to approximate $y = x^2$ in the Appendix B example.

$$\mathbf{W}_3 = \begin{bmatrix} 4.323 \\ -1.276 \\ -2.209 \end{bmatrix}. \tag{B13c}$$

and bias vectors

$$\mathbf{B}_1 = \begin{bmatrix} 2.424 & -0.203 & -2.053 \end{bmatrix}, \tag{B14a}$$

$$\mathbf{B}_2 = \begin{bmatrix} 0.520 & -1.258 & -0.828 \end{bmatrix}, \tag{B14b}$$

$$\mathbf{B}_3 = \begin{bmatrix} 0.588 \end{bmatrix}. \tag{B14c}$$

The final step is to calculate Eq. (B2) using the aforementioned equations, matrices, and vectors for the desired input x. To obtain the NNJ, let us calculate the analytical derivative through the MLP for a randomly generated input of $x = 0.812$. Solving for the derivative of the first hidden layer, Eq. (B10), we have

$$\frac{d\mathbf{y}_1}{d\mathbf{y}_0} = \begin{bmatrix} 1.735 \\ -0.850 \\ 1.404 \end{bmatrix}^\top \odot F_1'\left(0.812 \begin{bmatrix} 1.735 \\ -0.850 \\ 1.404 \end{bmatrix}^\top + \begin{bmatrix} 2.424 \\ -0.203 \\ -2.053 \end{bmatrix}^\top\right) \tag{B15a}$$

$$\frac{d\mathbf{y}_1}{d\mathbf{y}_0} = \begin{bmatrix} 1.735 \\ -0.850 \\ 1.404 \end{bmatrix}^\top \odot F_1'\left(\begin{bmatrix} 3.833 \\ -0.893 \\ 0.913 \end{bmatrix}^\top\right) \tag{B15b}$$

$$\frac{d\mathbf{y}_1}{d\mathbf{y}_0} = \begin{bmatrix} 1.735 \\ -0.850 \\ 1.404 \end{bmatrix}^\top \odot \begin{bmatrix} 0.021 \\ 0.206 \\ 0.204 \end{bmatrix}^\top \tag{B15c}$$

$$\frac{d\mathbf{y}_1}{d\mathbf{y}_0} = \begin{bmatrix} 0.036 & -0.175 & 0.287 \end{bmatrix}. \tag{B15d}$$

Note that the transpose of a row vector is used to save space. Likewise, solving for the derivative of the second hidden layer, Eq. (B11), we have

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} \odot F_2'\Bigg($$

$$F_1\left(\begin{bmatrix} 1.735 \\ -0.850 \\ 1.404 \end{bmatrix}^\top 0.812 + \begin{bmatrix} 2.424 \\ -0.203 \\ -2.053 \end{bmatrix}^\top\right) \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix}$$

$$+ \begin{bmatrix} 0.520 \\ -1.258 \\ -0.828 \end{bmatrix}^\top \Bigg) \quad \text{(B16a)}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} \odot F_2' \Bigg(F_1 \left(\begin{bmatrix} 3.833 \\ -0.893 \\ 0.913 \end{bmatrix}^\top \right) \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} + \begin{bmatrix} 0.520 \\ -1.258 \\ -0.828 \end{bmatrix}^\top \Bigg) \quad \text{(B16b)}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} \odot F_2' \Bigg(\begin{bmatrix} 0.979 \\ -0.290 \\ 0.286 \end{bmatrix}^\top \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} + \begin{bmatrix} 0.520 \\ -1.258 \\ -0.828 \end{bmatrix}^\top \Bigg) \quad \text{(B16c)}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} \odot F_2' \left(\begin{bmatrix} -0.134 \\ 0.711 \\ -0.022 \end{bmatrix}^\top \right) \quad \text{(B16d)}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \begin{bmatrix} -2.257 & 3.355 & 2.304 \\ 1.418 & -0.558 & -1.067 \\ 3.993 & -4.024 & -3.978 \end{bmatrix} \odot \begin{bmatrix} 0.249 \\ 0.221 \\ 0.250 \end{bmatrix}^\top \quad \text{(B16e)}$$

$$\frac{d\mathbf{y}_2}{d\mathbf{y}_1} = \begin{bmatrix} -0.562 & 0.741 & 0.576 \\ 0.353 & -0.123 & -0.267 \\ 0.994 & -0.889 & -0.994 \end{bmatrix}. \quad \text{(B16f)}$$

The derivative of the third output layer, Eq. (B12) is simply

$$\frac{d\mathbf{y}_3}{d\mathbf{y}_2} = \begin{bmatrix} 4.323 \\ -1.276 \\ -2.209 \end{bmatrix}. \quad \text{(B17)}$$

Lastly, if we substitute Eqs. (B15d), (B16f), and (B17) into Eq. (B2) we can calculate the NNJ as

$$\frac{d\mathbf{y}_3}{d\mathbf{x}} = \begin{bmatrix} 0.036 \\ -0.175 \\ 0.287 \end{bmatrix}^\top \begin{bmatrix} -0.562 & 0.741 & 0.576 \\ 0.353 & -0.123 & -0.267 \\ 0.994 & -0.889 & -0.994 \end{bmatrix} \begin{bmatrix} 4.323 \\ -1.276 \\ -2.209 \end{bmatrix}, \quad \text{(B18a)}$$

$$\frac{d\mathbf{y}_3}{d\mathbf{x}} = \begin{bmatrix} 0.203 & -0.207 & 0.218 \end{bmatrix} \begin{bmatrix} 4.323 \\ -1.276 \\ -2.209 \end{bmatrix}, \quad \text{(B18b)}$$

$$\frac{d\mathbf{y}_3}{d\mathbf{x}} = 1.623. \quad \text{(B18c)}$$

Comparing this result with the ground truth derivative of Eq. (B1):

$$y' = 2(0.812) = 1.624 \quad \text{(B19)}$$

we observe that our Jacobian calculations had a 0.062% error.

References

[1] Das, L., Sivaram, A., and Venkatasubramanian, V., 2020, "Hidden representations in deep neural networks: Part 2. Regression problems," Computers & Chemical Engineering, **139**, p. 106895.

[2] Buntine, W. L. and Weigend, A. S., 1994, "Computing second derivatives in feed-forward networks: A review," IEEE transactions on Neural Networks, **5**(3), pp. 480–488.

[3] Iserles, A., 2009, *A first course in the numerical analysis of differential equations*, Cambridge university press.

[4] Martins, J. R., Sturdza, P., and Alonso, J. J., 2003, "The complex-step derivative approximation," ACM Transactions on Mathematical Software (TOMS), **29**(3), pp. 245–262.

[5] Margossian, C. C., 2019, "A review of automatic differentiation and its efficient implementation," Wiley interdisciplinary reviews: data mining and knowledge discovery, **9**(4), p. e1305.

[6] Rozvany, G. I. N., 2009, "A critical review of established methods of structural topology optimization," Structural and Multidisciplinary Optimization, **37**(3), pp. 217–237.

[7] Kissel, M. and Diepold, K., 2020, "Sobolev Training with Approximated Derivatives for Black-Box Function Regression with Neural Networks," *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Vol. pt. II, Springer International Publishing, Conference Proceedings, Sep 19-20, 2019, pp. 399–414, doi: 10.1007/978-3-030-46147-8_24.

[8] Avrutskiy, V. I., 2021, "Enhancing Function Approximation Abilities of Neural Networks by Training Derivatives," IEEE Transactions on Neural Networks and Learning Systems, **32**(2), pp. 916–24.

[9] Kiran, R. and Naik, D. L., 2021, "Novel sensitivity method for evaluating the first derivative of the feed-forward neural network outputs," Journal of Big Data, **8**(1), pp. 1–13.

[10] Ledesma, S., Almanza-Ojeda, D.-L., Ibarra-Manzano, M.-A., Yepez, E. C., Avina-Cervantes, J. G., and Fallavollita, P., 2020, "Differential neural networks (DNN)," IEEE Access, **8**, pp. 156530–156538.

[11] Rodini, S., 2022, "Analytical derivatives of neural networks," Computer Physics Communications, **270**, p. 108169.

[12] Deng, H. and To, A. C., 2021, "A Parametric Level Set Method for Topology Optimization Based on Deep Neural Network," Journal of Mechanical Design, **143**(9), pp. 1–9.

[13] Shu, D., Cunningham, J., Stump, G., Miller, S. W., Yukish, M. A., Simpson, T. W., and Tucker, C. S., 2019, "3D Design Using Generative Adversarial Networks and Physics-Based Validation," Journal of Mechanical Design, **142**(7), pp. 1–15.

[14] Woldseth, R. V., Aage, N., Bærentzen, J. A., and Sigmund, O., 2022, "On the use of artificial neural networks in topology optimisation," Structural and Multidisciplinary Optimization, **65**(10), p. 294.

[15] Takahashi, Y., Suzuki, Y., and Todoroki, A., 2019, "Convolutional Neural Network-based Topology Optimization (CNN-TO) By Estimating Sensitivity of Compliance from Material Distribution," https://arxiv.org/abs/2001.00635, pp. 1–12.

[16] Watts, S., Arrighi, W., Kudo, J., Tortorelli, D. A., and White, D. A., 2019, "Simple, accurate surrogate models of the elastic response of three-dimensional open truss micro-architectures with applications to multiscale topology design," Structural and Multidisciplinary Optimization, **60**(5), pp. 1887–1920.

[17] Zhang, Y., Li, H., Xiao, M., Gao, L., Chu, S., and Zhang, J. H., 2019, "Concurrent topology optimization for cellular structures with nonuniform microstructures based on the kriging metamodel," Structural and Multidisciplinary Optimization, **59**(4), pp. 1273–1299.

[18] Zhang, Y., Gao, L., and Xiao, M., 2020, "Maximizing natural frequencies of inhomogeneous cellular structures by Kriging-assisted multiscale topology optimization," Computers & Structures, **230**, pp. 1–24.

[19] Zhang, Y., Xiao, M., Gao, L., Gao, J., and Li, H., 2020, "Multiscale topology optimization for minimizing frequency responses of cellular composites with connectable graded microstructures," Mechanical Systems and Signal Processing, **135**, p. 106369 (32 pp.).

[20] Bendsøe, M. P. and Kikuchi, N., 1988, "Generating optimal topologies in structural design using a homogenization method," Computer Methods in Applied Mechanics and Engineering, **71**(2), pp. 197–224.

[21] Pantz, O. and Trabelsi, K., 2008, "A post-treatment of the homogenization method for shape optimization," Siam Journal on Control and Optimization, **47**(3), pp. 1380–1398.

[22] Groen, J. P. and Sigmund, O., 2018, "Homogenization-based topology optimization for high-resolution manufacturable microstructures," International Journal for Numerical Methods in Engineering, **113**(8), pp. 1148–1163.

[23] Allaire, G., Geoffroy-Donders, P., and Pantz, O., 2019, "Topology optimization of modulated and oriented periodic microstructures by the homogenization method," Computers & Mathematics with Applications, **78**(7), pp. 2197–2229.

[24] Zhu, Y. C., Li, S. S., Du, Z. L., Liu, C., Guo, X., and Zhang, W. S., 2019, "A novel asymptotic-analysis-based homogenisation approach towards fast design of infill graded microstructures," Journal of the Mechanics and Physics of Solids, **124**, pp. 612–633.

[25] Li, D. W., Dai, N., Tang, Y. L., Dong, G. Y., and Zhao, Y. F., 2019, "Design and Optimization of Graded Cellular Structures With Triply Periodic Level Surface-Based Topological Shapes," Journal of Mechanical Design, **141**(7), pp. 1–13.

[26] Wang, C., Zhu, J. H., Zhang, W. H., Li, S. Y., and Kong, J., 2018, "Concurrent topology optimization design of structures and non-uniform parameterized lattice microstructures," Structural and Multidisciplinary Optimization, **58**(1), pp. 35–50.

[27] Imediegwu, C., Murphy, R., Hewson, R., and Santer, M., 2019, "Multiscale structural optimization towards three-dimensional printable structures," Structural and Multidisciplinary Optimization, **60**(2), pp. 513–525.

[28] White, D. A., Arrighi, W. J., Kudo, J., and Watts, S. E., 2019, "Multiscale topology optimization using neural network surrogate models," Computer Methods in Applied Mechanics and Engineering, **346**, pp. 1118–1135.

[29] Black, N. and Najafi, A. R., 2023, "Deep neural networks for parameterized homogenization in concurrent multiscale structural optimization," Structural and Multidisciplinary Optimization, **66**(1), p. 20.

[30] Zheng, L., Kumar, S., and Kochmann, D. M., 2021, "Data-driven topology optimization of spinodoid metamaterials with seamlessly tunable anisotropy," Computer Methods in Applied Mechanics and Engineering, **383**, p. 113894.

[31] Rastegarzadeh, S., Wang, J., and Huang, J., 2023, "Neural Network-Assisted Design: A Study of Multiscale Topology Optimization With Smoothly Graded Cellular Structures," Journal of Mechanical Design, **145**(1), p. 011701.

[32] Rozvany, G. I. N., Zhou, M., and Birker, T., 1992, "Generalized Shape Optimization without Homogenization," Structural Optimization, **4**(3-4), pp. 250–252.

[33] Andreassen, E. and Andreasen, C. S., 2014, "How to determine composite material properties using numerical homogenization," Computational Materials Science, **83**, pp. 488–495.

[34] McClure, N., 2017, *TensorFlow machine learning cookbook*, PACKT publishing Ltd.

[35] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E., 2017, "A survey of deep neural network architectures and their applications," Neurocomputing, **234**, pp. 11–26.

[36] Kaup, L. and Kaup, B., 2011, *Holomorphic functions of several variables: an introduction to the fundamental theory*, Vol. 3, Walter de Gruyter.

[37] Squire, W. and Trapp, G., 1998, "Using complex variables to estimate derivatives of real functions," SIAM review, **40**(1), pp. 110–112.

[38] Neidinger, R. D., 2010, "Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming," Siam Review, **52**(3), pp. 545–563.

[39] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., and Isard, M., 2016, "TensorFlow: A System for Large-Scale Machine Learning," *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, Conference Proceedings, Nov 2-4, 2016, pp. 265–283, https://dl.acm.org/doi/proceedings/10.5555/3026877

[40] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., and Bright, J., 2020, "SciPy 1.0: fundamental algorithms for scientific computing in Python," Nature methods, **17**(3), pp. 261–272.

[41] Sigmund, O., 1997, "On the design of compliant mechanisms using topology optimization," Mechanics of Structures and Machines, **25**(4), pp. 493–524.

[42] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A., 2017, "Hyperband: A novel bandit-based approach to hyperparameter optimization," The Journal of Machine Learning Research, **18**(1), pp. 6765–6816.

[43] Ruder, S., 2016, "An overview of gradient descent optimization algorithms," https://arxiv.org/abs/1609.04747, pp. 1–14.

[44] Najmon, J. and Tovar, A., 2023, "Comparing Derivatives of Neural Networks for Regression," *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE*, American Society of Mechanical Engineers, Conference Proceedings, Aug 20-23, 2023, pp. 1–7.