Composite Structures 272 (2021) 114137

Contents lists available at ScienceDirect

Composite Structures

journal homepage: www.elsevier.com/locate/compstruct

Learning composite constitutive laws via coupling Abaqus and deep neural network

Fei Tao *, Xin Liu, Haodong Du, Wenbin Yu

School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN 47907-2045, USA

ARTICLE INFO

Keywords:
Composite
Constitutive Law
Deep neural network
Abaqus

ABSTRACT

The commercial finite element (FE) code Abaqus is coupled with the deep neural network (DNN) model, namely Abaqus-DNN mechanics system, to learn the constitutive law of the fiber-reinforced composite. The proposed system enables data communication between Abaqus and DNN model, which leverages the versatile FE analysis ability of Abaqus and the powerful machine learning using DNN. Abaqus-DNN enables DNN to learn the constitutive law in a form-free manner. The learned result automatically satisfies the equilibrium and kinematics equations, which avoids inaccuracies associated with the presumed functions in the constitutive laws and guaranteed the learned constitutive law following the laws of physics. The Abaqus-DNN mechanics system was implemented to learn the full set of engineering constants of the constituents of a fiber-reinforced composite. Furthermore, the proposed system was applied to learn the progressive damage constitutive law of a fiber-reinforced composite laminate. The backward propagation equations of the neural network were modified to track the gradient of the loss function from Abaqus to DNN. The results show that Abaqus-DNN can accurately learn constitutive laws based on structural level data. This system provides a generalized approach for learning unknown physics inside a mechanics system by coupling neural network with commercial finite element codes.

1. Introduction

Composite materials have been widely used in various industries, such as aerospace, automobile, and wind turbines. Due to their high strength/stiffness-to-weight ratio and excellent resistance to fatigue, composite materials are particularly attractive to advanced structures, e.g. aircraft and wind turbine blades. Although the material properties of composites are desirable, the mechanical behaviors of composites are very complex, as composites are heterogeneous and fragile to damage during manufacturing and in-service [1]. Extensive research has been conducted focusing on modeling the constitutive behavior of composites. For example, the multiscale modeling, such as representative volume element (RVE) [2] or mechanics structure genome (MSG) [3,4], has been proposed to compute effective constitutive relations. The continuum damage mechanics (CDM) is developed to model the composites damage initiation and evolution. The CDM assumes that the material degradation caused by micro-defects can be described by a set of continuous variables [5–9]. Although many advances have been made, it is still difficult and expensive to model the constitutive law of a composite accurately. This is mainly due to the heterogeneity, anisotropy, and complicated damage mechanisms of the composite material [10].

Recently, data-driven method has received rapidly increasing interest in the constitutive modeling [11,12]. Deep neural network (DNN) is one of the most popular method in the data-driven constitutive modeling. One attraction of DNN is that it can build a complex nonlinear relationship in a form-free manner which avoids the possible inaccuracies associated with the presumed functions in the constitutive laws [13]. DNN has the potential to reduce the computational cost and discover unknown constitutive laws. Currently, the study of DNN in constitutive modeling mainly focuses on reducing computational cost. These researches used direct data, such as strain–stress pairs or strain-energy pairs, to train a surrogate model to replace the computation expensive process in a physics-based simulation. For example, Ghaboussi et al. used a knowledge-based neural network to model concrete constitutive behavior [14]. Shen et al. developed a hyperelastic model based on neural network and implemented it in Abaqus [15]. Wang and Sun utilized a neural network to learn the plasticity behavior of a porous material [16]. Mozaffar et al. implemented DNN to predict the path-dependent plasticity [17]. The neural network models trained by these studies are in good agreement with the training data. However, these neural networks cannot fully capture the constitutive features of the material, since the source data for these studies are

* Corresponding author.
E-mail address: tfei@purdue.edu (F. Tao).

https://doi.org/10.1016/j.compstruct.2021.114137
Received 21 August 2020; Revised 30 April 2021; Accepted 17 May 2021
Available online 24 May 2021

either generated by numerical simulations or by simple mechanical tests. The data generated by numerical simulations is biased, as it requires a known constitutive law with certain assumptions. In addition, the simple mechanical tests cannot directly measure the heterogeneous strain–stress data or the measured strain–stress data is uniform along one direction and cannot provide enough information for high dimensional, i.e., two-dimensional (2D) or three-dimensional (3D), constitutive laws.

To enable DNN to learn unknown constitutive laws with unbiased, heterogeneous strain–stress data, some researchers proposed to use inverse modeling to determine the inputs and outputs of DNN based on indirect data, such as forces and displacements which can be directly measured from experiments. For example, Ghaboussi et al. proposed an auto-progressive method to train neural network to learn the constitutive law of concrete based on the experimental load–deflection data [18]. However, this method is inefficient due to two FE simulations are needed to compute the stresses and strains, respectively. Huang and Xu presented a method to use DNN to replace the constitutive law in the FE model, which enables DNN to learn the constitutive law based on the experimental measurement [19,20]. This method just needs to solve the FE model one time during each training iteration. But this method requires a full-field observation of the forces and displacements, while only partially observed data (data from limited measuring points) is usually available in most experiments. The full-field observation requirement limits the application of this method to complex problems, such as a 3D solid problem. Liu. et al. developed a method to couple FE analysis with neural network to form a coupled mechanics system so that the input and output of the neural network can be determined at each incremental step [21,22]. Liu's method allows neural network to discover the constitutive law with a partial observation of forces and displacements. These proposed methods have been applied to different materials and obtained good nonlinear constitutive models. Nevertheless, these methods need to write a FE code based on an automatic differentiation package, such as Tensorflow or PyTorch. Due to the limitation of the automatic differentiation packages, the FE model assembling and solving process is significantly time-consuming. In addition, these methods require writing in–house codes to generate FE model geometry and mesh, which makes it hard to apply to complex nonlinear FE problems.

In this paper, DNN was integrated with Abaqus to form a coupled mechanics system, denoted as Abaqus-DNN. The proposed approach enables data communication between Abaqus and DNN and makes it possible to use partially observed data to train DNN. This approach leverages the versatile FE analysis ability of Abaqus and the powerful machine learning using DNN. Two examples were studied using the proposed approach in this paper. A 3D fiber-reinforced composite was presented to learn the full set of engineering constants of fiber and matrix. This example is mainly used to demonstrate that the Abaqus-DNN mechanics system has the potential to determine Young's modulus of fiber in the x_2 direction, which is usually difficult to measure from experiments. The second example used a plane stress model to learn an Abaqus build-in progressive damage constitutive law of a fiber-reinforced laminate. This example is to demonstrate that the Abaqus-DNN mechanics system can learn the nonlinear constitutive law based on indirect data in a form-free manner, which has the potential to improve the accuracy of the nonlinear constitutive laws. The reason is that many constitutive laws were constructed based on the homogenized behavior of composites using a postulated function [9,23,24]. However, the choice of the presumed functions is based on the experiences with various assumptions, which may cause a loss of accuracy in the solution.

The remainder of this paper is organized as follows. The fundamentals of DNN and the Abaqus-DNN mechanics system are introduced in Sections 2 and 3 respectively. Then, we applied the Abaqus-DNN mechanics system to determine the elastic constants of constituents in a fiber-reinforced composite and nonlinear progressive damage constitutive law in Section 4. The discussion of the two problem results is presented in Section 5. Finally, the conclusion is given in Section 6.

2. Fundamentals of deep neural network

Artificial neural networks are computing systems inspired by neuroscience. The basic idea of the neural network is to use multiple neurons to describe a complex system. In fact, a neural network model is a framework where various machine learning algorithms work simultaneously to process complex data sets [25].

As shown in Fig. 1, a neural network is usually composed of an input layer, one or more hidden layer(s), and an output layer. The input layer is the first layer of the neural network which takes input values and passes them to the next layer. The input layer does not apply any operation on the input values, since the input layer has no weights and biases. The hidden layers have neurons that apply different operations to the input data. All the neurons in a hidden layer are connected to each neuron in the previous layer and the next layer. Thus, the hidden layers are fully connected. An artificial neural network is called a deep neural network (DNN) if the number of hidden layers is larger than two. The output layer is the last layer of the network which gives the predicted values of the model [26].

The connections between neurons are built by weights (w_{ij}) and biases (b_j). The weights are used to measure the strength of the connection between neurons, and the biases are used to tune the activation function [27]. The outputs from preceding neurons (a_i) will multiply the corresponding weights. Then summing these values and adding the bias of that neuron gives

$$z_j = \sum_{i=1}^{n} a_i w_{ij} + b_j \tag{1}$$

The z_j will be passed onto the activation function, which is a nonlinear function and results in a scalar value for the neuron. The scalar will be the input of the following layer. In this paper, the choice of the activation function is the rectified linear unit function (ReLU) [27].

Neural network is usually trained with a stochastic gradient descent method [28]. Thus, an objective loss function needs to be defined. In current context, the loss function is expressed as the mean square error (MSE):

$$L_a = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 \tag{2}$$

where n is the number of training samples, $y^{(i)}$ is the predicted result by the neural network model and $\hat{y}^{(i)}$ is the exact result of the training

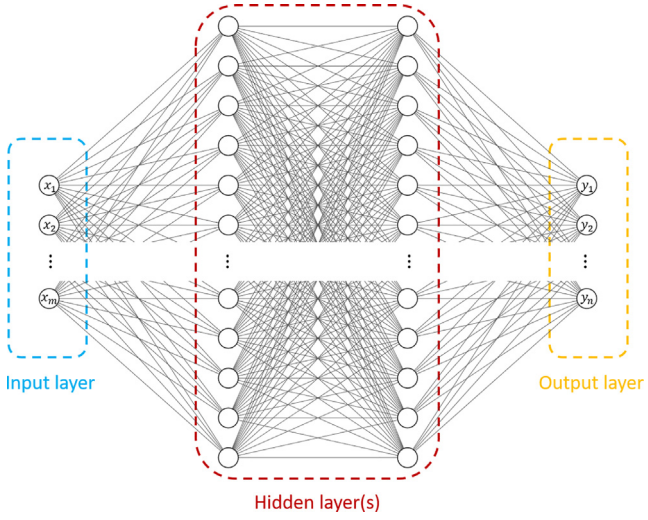


Fig. 1. The structure of an artificial neural network.

samples. To find the minimum of the loss function, the weights and biases can be updated using the following equations:

$$w'_{ij} = w_{ij} - \eta \frac{\partial L_a}{\partial w_{ij}} \tag{3a}$$

$$b'_j = b_j - \eta \frac{\partial L_a}{\partial b_j} \tag{3b}$$

where w'_{ij} and b'_j are the corresponding new weights and biases, $w_{ij}$and b_j are the old weights and biases, η is the learning rate which is a hyper-parameter that dominates the amount by which the model is to be changed in response to the estimated error each time the model weights and biases are updated [28]. The loss gradients in Eq. (3) can be determined with the backward propagation equations. The backward propagation equations are:

$$\boldsymbol{\delta}^L = (\boldsymbol{a}^L - \boldsymbol{y}) \odot g'(\boldsymbol{z}^L) \tag{4a}$$

$$\boldsymbol{\delta}^l = \left((\boldsymbol{w}^{l+1})^T \boldsymbol{\delta}^{l+1} \right) \odot g'(\boldsymbol{z}^l) \tag{4b}$$

$$\frac{\partial L_a}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{4c}$$

$$\frac{\partial L_a}{\partial b^l_j} = \delta^l_j \tag{4d}$$

where δ is the error of the layer, the superscript L denotes the output layer and l denotes the hidden layers, $g(\cdot)$ is the activation function. The bold symbols in Eq. (4) indicate the variables are of the matrix form. The $\odot$ represents the element-wise product of two vectors. The errors defined in Eqs. (4a) and (4b) are used to simplify Eqs. (4c) and (4d). For example, for a neural network with two hidden layers, the weight of the first neuron of the first hidden layer to the first neuron of the second hidden layer (w^2_{11}) can be updated:

$$\frac{\partial L_a}{\partial w^2_{11}} = \frac{\partial L_a}{\partial y} \frac{\partial y}{\partial z^L} \frac{\partial z^L}{\partial a^2_1} \frac{\partial a^2_1}{\partial z^2_1} \frac{\partial z^2_1}{\partial w^2_{11}} = \delta^3 w^3_{11} g'(z^2_1) a^1_1 = \delta^2_1 a^1_1 \tag{5}$$

Eq. (5) can be derived from Eq. (4c) directly.

3. Abaqus coupled deep neural network mechanics system

3.1. The framework of Abaqus-DNN mechanics system

The Abaqus-DNN mechanics system enables data communication between Abaqus and DNN. The proposed system can be described using Fig. 2. The $\boldsymbol{I}^s$ and $\boldsymbol{O}^s$ represent the system input and output. The system input and output can be directly measured from experiments, such as the displacement and force in a tensile test. The Abaqus in Fig. 2 refers to the solving process of the FE model. The Subsystem in the figure is used (or combing with Abaqus) to derive the input i_k for DNN. The choice of the Subsystem is problem dependent. It can be an analytical equation, a filter, or can be ignored for some scenarios. The output of the DNN model are represented by the o_l. The DNN output o_l will contribute to the system output $\boldsymbol{O}^s$ which will be used to construct the loss function. The loss function is the objective function and will be minimized through an optimizer. The selection of i_k and o_l are based on the needs of the problem. For example, if one wants to train a relationship between strain and stress, one can set the i_k and o_l to be strain and stress, respectively. Sometimes, if it is more convenient to build a relationship between strain and the tangent stiffness, then one can set the i_k and o_l to be strain and independent variables of the tangent stiffness, respectively.

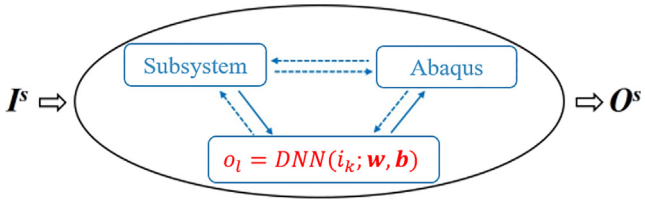


Fig. 2. The framework of the Abaqus-DNN mechanics system.

For this mechanics system, the simplest scenario is an uncoupled problem represented by the solid blue line in the figure. The input data (i_k) of the DNN model can be obtained directly from the system input ($\boldsymbol{I}^s$) through Subsystem. The outputs of the DNN model (o_l) are sent to Abaqus to compute the system output ($\boldsymbol{O}^s$). For example, when applying the proposed approach to discover the linear elastic constants of a material, the output of the DNN shall be the engineering constants. Since the engineering constants (o_l) is independent of DNN input (i_k), this system becomes an uncoupled system.

A more complicated and interesting scenario for this system is to deal with a fully coupled problem. For example, for nonlinear problems, the constitutive law is expressed as the mapping between strains (input) and stresses (output) via DNN. The strains have to be derived using the system input with Subsystem, Abaqus, and DNN, as solving the strains requires using the constitutive equations that are represented by the DNN model. Thus, the determining of DNN input also depends on the output of DNN. The DNN model is fully coupled with the entire mechanics system.

The proposed system has several benefits. Firstly, the proposed system enables data communication between Abaqus and neural network. This facilitates the creating of the FE model by using powerful commercial codes and avoids rewriting the FE solver based on the automatic differentiation package. Secondly, this system enables DNN to learn a constitutive law based on indirect data, which can be easily measured from experiments. Thirdly, the constitutive law is learned in a form-free manner that avoids the accuracy loss caused by the presumed function forms of constitutive laws. Finally, since FE is coupled to compute the DNN outputs, the learned outputs automatically satisfy equilibrium and the kinematics equations, which guaranteed that the learned result follows the law of physics.

A two-round training scheme was proposed to train the Abaqus-DNN mechanics system. The training process is shown in Fig. 3. The purpose of the first-round training is to determine accurate inputs and outputs from indirect experiment data. The first-round training will be performed incrementally. The experimental data, i.e. the load and corresponding displacements, will be divided into n sets, with n as the number of the total training steps of the first-round. The choice of n needs to consider convergence and performance of the entire system. As a too small n will make the training inefficient, a too large n may cause the neural network to fail to converge. At each step, the initialized weights, biases, and neural network inputs are obtained from the previous step. The load and DNN predicted constitutive law will be transferred into the Abaqus. Abaqus will solve the FE model and compute the gradient of the displacement with respect to the constitutive law parameters via design sensitivity analysis (DSA) [29]. Next, the difference between the observed displacements and computed displacements will be compared. If the difference does not reach the termination criterion, the weights and biases will be updated through an optimizer. Otherwise, the inputs, outputs, and model parameters of DNN will be saved for the second-round training and transferred to the next step as initial parameters. By repeating doing that for all the datasets, the complete sets of inputs and outputs of DNN can be obtained. After completing the first-round training, the second-round training will be performed to approximate the overall relationship between the accurate inputs and outputs.

3.2. Derivation of modified backward propagation equations

To update weights and biases of DNN in the system, the backward propagation equations need to be derived. Expressing the constitutive law as

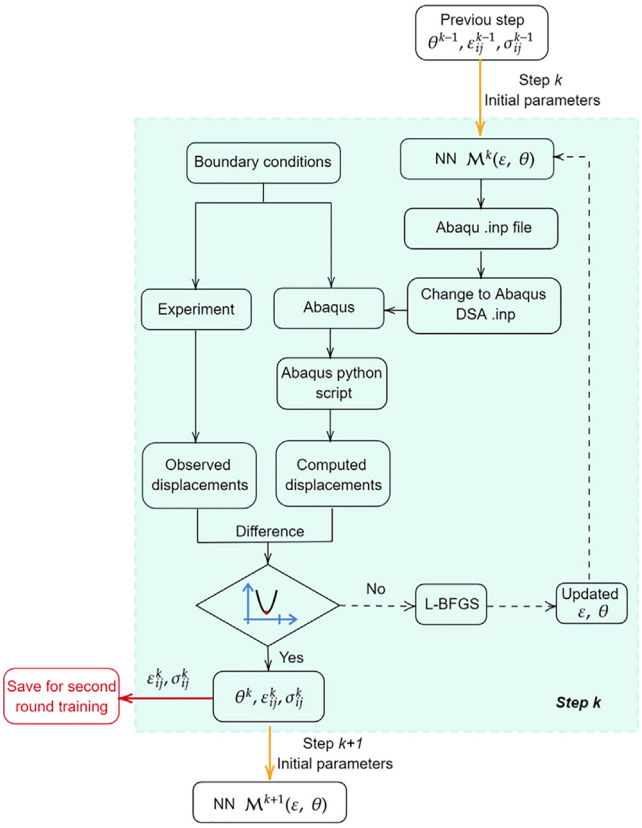


Fig. 3. The flowchart of the training of the Abaqus-DNN mechanics system.

$$\boldsymbol{\sigma} = \mathscr{M}(\boldsymbol{\varepsilon}; \boldsymbol{\theta}) \tag{6}$$

where $\mathscr{M}$ is the neural network model, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\sigma}$ are strains and stresses, $\boldsymbol{\theta}$ represent weights and biases. The measurable data from experiments are forces and displacements $(\boldsymbol{F}_i, \hat{\boldsymbol{u}}_i)$. Then the loss function $\boldsymbol{L}(\boldsymbol{\theta})$ can be formed as

$$\boldsymbol{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i}^{N}(\boldsymbol{u}_i(\boldsymbol{F}_i, \mathscr{M}(\boldsymbol{\varepsilon}; \boldsymbol{\theta})) - \hat{\boldsymbol{u}}_i)^2 \tag{7}$$

where $\boldsymbol{u}_i(\boldsymbol{F}_i, \mathscr{M}(\boldsymbol{\varepsilon}; \boldsymbol{\theta})))$ and $\hat{\boldsymbol{u}}$ are the computed displacement and observed displacement respectively, N is the total number of measurable datasets. The optimization of the loss function $\boldsymbol{L}$ is to find a set of weights and biases $\boldsymbol{\theta}$ of the neural network that minimize the loss function. By the chain rule, the parameters of the neural network model can be updated as

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\theta}} = \frac{1}{N}\sum_{i=1}^{N} 2(\boldsymbol{u}_i - \hat{\boldsymbol{u}}_i)\left(\sum_{m=1}^{M}\frac{\partial \boldsymbol{u}_i}{\partial \mathscr{M}_m}\frac{\partial \mathscr{M}_m}{\partial \boldsymbol{\theta}}\right) \tag{8}$$

where M is the total number of outputs of DNN that contributed to system output, $\mathscr{M}_m$ is the mth DNN output that contributed to the system output. In Eq. (8), $(\boldsymbol{u}_i - \hat{\boldsymbol{u}}_i)$ can be calculated analytically, $\partial \boldsymbol{u}_i / \partial \mathscr{M}_m$can be evaluated using the Abaqus design sensitivity analysis capability, and $\partial \mathscr{M}_m / \partial \boldsymbol{\theta}$ can be handled by the automatic differentiation package.

The key to enable data communication between Abaqus and DNN is to form Eq. (8) correctly. However, Eq. (8) can be very lengthy under certain conditions. For example, for a 3D nonlinear FE model that has 100 reduced integration elements. If the model is made of a transversely isotropic material which has five independent engineering constants, then, for each training iteration, every observed node needs to compute the sensitivity of the displacement at that node in terms of 500 material parameters. The detailed explanation of the number of material parameters of the nonlinear model is presented in Section 4. Considering a converged FE model can easily exceed 100 elements and the training needs multiple observation data points, Eq. (8) can be very lengthy. To ensure that the forming of Eq. (8) is correct, the explicit form of the backward propagation equation will be derived in Section 4.

The previous studies have shown that the second-order optimizer works better for the FE coupled neural network problems [19,20,22]. In this work, the Limited-memory BFGS (L-BFGS) optimizer [30] was used to train the Abaqus-DNN model. For a FE code written by an automatic differentiation package, the optimization only needs to focus on the forward propagation. The backward propagation can be handled by the automatic differentiation package. However, for Abaqus-DNN, the computation of backward propagation of the DNN and FE model has to be done separately. Therefore, it is necessary to connect the backward propagation between DNN and FE model following Eq. (8). The connected equation will be transferred to L-BFGS to update DNN parameters.

4. Applications

This section presented two examples using the Abaqus-DNN mechanics system. The first example applied Abaqus-DNN to learn the full set of engineering constants of the constituents of a fiber-reinforced composite. This example can be viewed as a traditional inverse problem, since it only needs to determine several independent constants in the stiffness matrix. It is not necessary to use DNN to approximate the stiffness matrix. However, the algorithm remains the same to integrate Abaqus with DNN to deal with linear and nonlinear problems. Therefore, this problem is used for demonstration purpose. The input dimension of the neural network of this problem was set to be one and the value of the input was fixed to be one. The second problem implemented Abaqus-DNN to learn a composite laminate progressive damage law. In both problems, the training data are generated numerically. It is noted that all the observation data in this paper refer to the results from the simulations with the constitutive law to be learned by the DNN model.

4.1. Learning linear elastic constants of constituents of a fiber-reinforced composite

A 3D square packed model is used to model the linear elastic behavior of a fiber-reinforced composite as shown in Fig. 4. The model has $L \times W \times H = 1 \times 1 \times 1$ mm and is fixed on the $x = 0$ surface. The surface pressures are applied at $x = 1$ and $z = 1$ planes with magnitude $p = -10$ MPa respectively. This model has two constituents, which are fiber and matrix. The fiber volume fraction is 0.6. Both materials are assumed to be made of homogeneous linear elastic material. The fiber and matrix are assumed to be transversely isotropic and isotropic, respectively. The engineering constants of these two materials are listed in Table 1.

For a linear elastic material, the constitutive law is

$$\boldsymbol{\sigma} = \boldsymbol{C}\varepsilon \tag{9}$$

where $\boldsymbol{\sigma}$ is the stress, $\boldsymbol{\varepsilon}$ is the strain, $\boldsymbol{C}$ is the stiffness matrix. Eq. (9) can be replaced by a DNN as shown in Eq. (6). For a FE analysis, the Jacobian of $\boldsymbol{\sigma}(\varepsilon)$ is required to form the global stiffness matrix to solve for the unknown variables. Although DNN can form an accurate mapping between input and output, it is difficult to guarantee the accuracy of the gradient of input to output. Thus, instead of applying the DNN model to learn the mapping between stresses and strains directly, the DNN model is used to predict the Jacobian of $\boldsymbol{\sigma}(\varepsilon)$. The general form of the Jacobian of $\boldsymbol{\sigma}(\varepsilon)$ is

$$\mathscr{M}(\boldsymbol{\varepsilon}, \boldsymbol{\theta}) = \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}} = \begin{bmatrix} \frac{\partial \sigma_{11}}{\partial \varepsilon_{11}} & \frac{\partial \sigma_{11}}{\partial \varepsilon_{22}} & \frac{\partial \sigma_{11}}{\partial \varepsilon_{33}} & \frac{\partial \sigma_{11}}{\partial \gamma_{23}} & \frac{\partial \sigma_{11}}{\partial \gamma_{13}} & \frac{\partial \sigma_{11}}{\partial \gamma_{12}} \\ & \frac{\partial \sigma_{22}}{\partial \varepsilon_{22}} & \frac{\partial \sigma_{22}}{\partial \varepsilon_{33}} & \frac{\partial \sigma_{22}}{\partial \gamma_{23}} & \frac{\partial \sigma_{22}}{\partial \gamma_{13}} & \frac{\partial \sigma_{22}}{\partial \gamma_{12}} \\ & & \frac{\partial \sigma_{33}}{\partial \varepsilon_{33}} & \frac{\partial \sigma_{33}}{\partial \gamma_{23}} & \frac{\partial \sigma_{33}}{\partial \gamma_{13}} & \frac{\partial \sigma_{33}}{\partial \gamma_{12}} \\ & & & \frac{\partial \sigma_{23}}{\partial \gamma_{23}} & \frac{\partial \sigma_{23}}{\partial \gamma_{13}} & \frac{\partial \sigma_{23}}{\partial \gamma_{12}} \\ & \text{symmetric} & & & \frac{\partial \sigma_{13}}{\partial \gamma_{13}} & \frac{\partial \sigma_{13}}{\partial \gamma_{12}} \\ & & & & & \frac{\partial \sigma_{12}}{\partial \gamma_{12}} \end{bmatrix} \tag{10}$$

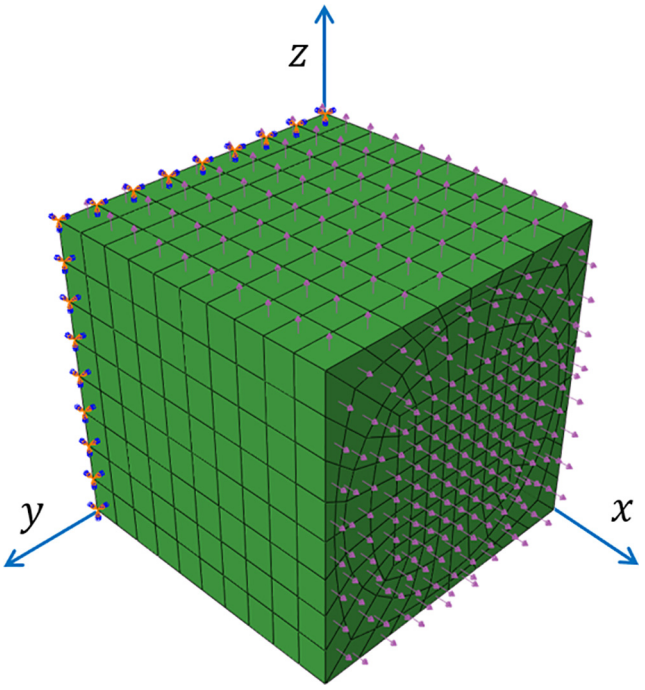


Fig. 4. A squared packed composites model.

For a linear elastic problem, $\mathscr{M}$ can be regarded as the linear elastic stiffness matrix $\boldsymbol{C}$. Clearly, $\boldsymbol{C}$ is not a function of $\boldsymbol{\varepsilon}$, as material properties of linear elastic material are constant. Thus, $\boldsymbol{C}(\boldsymbol{\varepsilon},\boldsymbol{\theta})$ can be reduced to $\boldsymbol{C}(\boldsymbol{\theta})$. This simplification makes the problem become an uncoupled problem as represented by the solid blue line in Fig. 2. Thus, the input of DNN can be set as a fixed single value. Besides, since the fiber and matrix are assumed to be transversely isotropic and isotropic respectively, Eq. (10) reduces to

$$\mathscr{M}(\boldsymbol{\theta})=\boldsymbol{C}(\boldsymbol{\theta})=\begin{bmatrix} C_{11}(\boldsymbol{\theta}) & C_{12}(\boldsymbol{\theta}) & C_{12}(\boldsymbol{\theta}) & 0 & 0 & 0 \\ & C_{22}(\boldsymbol{\theta}) & C_{23}(\boldsymbol{\theta}) & 0 & 0 & 0 \\ & & C_{22}(\boldsymbol{\theta}) & 0 & 0 & 0 \\ & & & \frac{C_{22}(\boldsymbol{\theta})-C_{23}(\boldsymbol{\theta})}{2} & 0 & 0 \\ & \text{symmetric} & & & C_{55}(\boldsymbol{\theta}) & 0 \\ & & & & & C_{55}(\boldsymbol{\theta}) \end{bmatrix} \tag{11}$$

It is needed to point out that the simplification in Eq. (11) is not a constraint for the Abaqus-DNN. It is possible to use a fully populated matrix if the mechanical property of the material is fully unknown or the extension and shear coupling does exist in the material.

After discretizing the computation domain and assembling the global stiffness matrix, the unknown displacements can be expressed as [31]

$$\boldsymbol{v}^h=\boldsymbol{K}(\boldsymbol{C})^{-1}\boldsymbol{F} \tag{12}$$

where $\boldsymbol{v}^h$ is the unknown displacement, $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{F}$ is the external force. Eq. (12) shows that the unknown displacement $\boldsymbol{v}^h$ is a function of $\boldsymbol{C}(\boldsymbol{\theta})$. Therefore, the structure of the Abaqus-DNN system can be described with Fig. 5. As shown in the figure, both fiber and matrix neural networks have one input. As mentioned earlier, the value of the input was set to be one. The fiber neural network has five outputs that account for the five independent engineering constants of a transversely isotropic material. The outputs of the matrix neural network are two which represent the independent engineering constants of an isotropic material.

The backward propagation equation can be explicitly expressed as

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\theta}} &= \frac{1}{N}\sum_{i=1}^{N}2(\boldsymbol{u}_i-\hat{\boldsymbol{u}}_i)\left(\sum_{i=1}^{M}\frac{\partial \boldsymbol{u}_i}{\partial \mathscr{M}_m}\frac{\partial \mathscr{M}_m}{\partial \boldsymbol{\theta}}\right) \\ &= \frac{2}{N}\left[(u_1-\hat{u}_1)\left(\frac{\partial u_1}{\partial y_{11}}\frac{\partial y_{11}}{\partial \boldsymbol{\theta}}\right)+(u_1-\hat{u}_1)\left(\frac{\partial u_1}{\partial y_{12}}\frac{\partial y_{12}}{\partial \boldsymbol{\theta}}\right)+\cdots+(u_1-\hat{u}_1)\left(\frac{\partial u_1}{\partial y_{22}}\frac{\partial y_{22}}{\partial \boldsymbol{\theta}}\right)\right] \\ &\quad+\frac{2}{N}\left[(u_2-\hat{u}_2)\left(\frac{\partial u_2}{\partial y_{11}}\frac{\partial y_{11}}{\partial \boldsymbol{\theta}}\right)+(u_2-\hat{u}_2)\left(\frac{\partial u_2}{\partial y_{12}}\frac{\partial y_{12}}{\partial \boldsymbol{\theta}}\right)+\cdots+(u_2-\hat{u}_2)\left(\frac{\partial u_2}{\partial y_{22}}\frac{\partial y_{22}}{\partial \boldsymbol{\theta}}\right)\right] \\ &\quad+\quad\vdots \\ &\quad\frac{2}{N}\left[(u_N-\hat{u}_N)\left(\frac{\partial u_N}{\partial y_{11}}\frac{\partial y_{11}}{\partial \boldsymbol{\theta}}\right)+(u_N-\hat{u}_N)\left(\frac{\partial u_N}{\partial y_{12}}\frac{\partial y_{12}}{\partial \boldsymbol{\theta}}\right)+\cdots+(u_N-\hat{u}_N)\left(\frac{\partial u_N}{\partial y_{22}}\frac{\partial y_{22}}{\partial \boldsymbol{\theta}}\right)\right]\end{aligned} \tag{13}$$

where $y_{1p}, p=1,\ldots,5$ and $y_{2q}, q=1,2$ are the outputs of the fiber neural network and matrix neural network respectively.

This model used 1920 C3D8R elements after a convergence study. After trial-and-error tests, the structure of the neural network for fiber was identified as $[1,10,10,5]$, which indicates that the neural network has one input, five outputs, and two hidden layers with each hidden layer have ten neurons. The activation function was chosen to be the ReLU function. For the matrix, the neural network was chosen to be $[1,10,10,2]$. The final step's load and the corresponding displacements were selected for training. Therefore, the training can be completed within one step as shown in Fig. 3. The observation nodes used for training are shown along the red line in Fig. 6, which has 16 nodes in total. Note that the training only used the nodes on the exterior surface. This indicates that the proposed Abaqus-DNN model can be completed using only the surface displacements, which are easily measured from an experiment.

4.2. Learning the damage constitutive law of a fiber-reinforced composite

As shown in Fig. 7, a plane stress model with $L\times H=2\times 1$inch is given. The center of the left surface $x=0$ is pinned with $u_x=u_y=0$. The other area along $x=0$ is constrained with $u_x=0$. A uniform displacement, which is applied using the Abaqus coupling function, is applied at the right surface at $x=2$ inch with magnitude $u_x=0.1$inch.

The material of the model is set to be the fiber-reinforced composite and exhibits a damage behavior. The material properties and damage parameters are listed in Table 2. The damage in the fiber-reinforced composite usually initiates without the large plastic deformation. Therefore, plasticity can be ignored in the analysis. In this paper, we applied Abaqus-DNN to learn Abaqus build-in anisotropic damage constitutive law for the fiber-reinforced composite [32,24,29]. For this constitutive law, the onset of damage is determined by the Hashin failure criterion, the damage evolution is based on the fracture energy dissipation during the damage process [33]. The response of the material can be computed

$$\boldsymbol{\sigma}=\boldsymbol{C}_d\boldsymbol{\varepsilon} \tag{14}$$

where $\boldsymbol{C}_d$ is the damaged stiffness matrix, which has the expression

Table 1
Fiber and matrix material properties.

	E_1 (GPa)	E_2 (GPa)	ν_{12}	G_{12} (GPa)	G_{23} (GPa)
Fiber	276.00	19.50	0.28	70.00	7.62
		E (GPa)		ν	
Matrix		4.76		0.36	

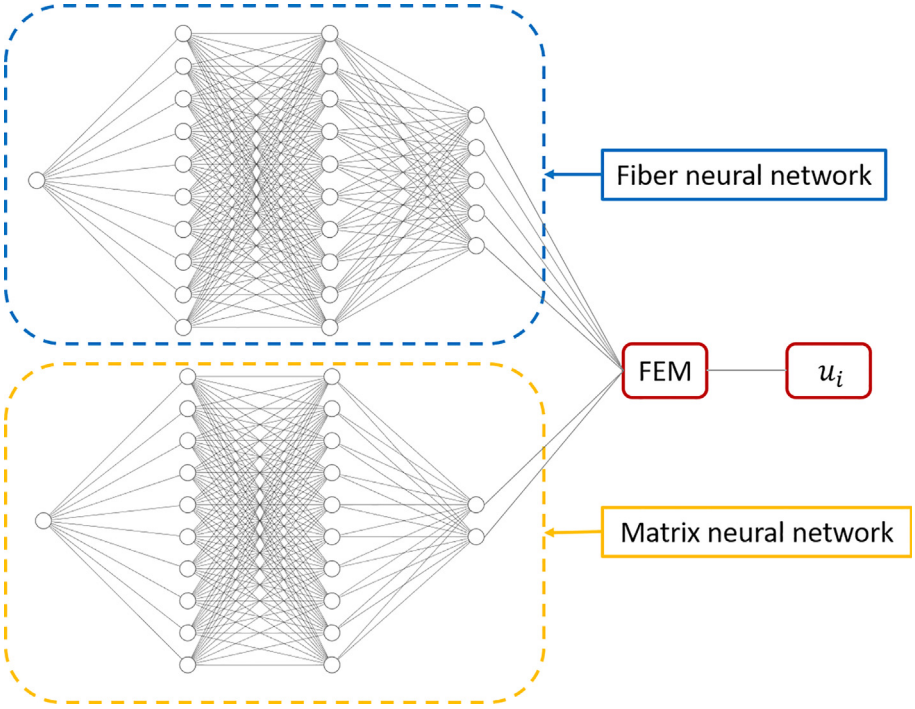


Fig. 5. The structure of Abaqus-DNN for a linear elastic problem.

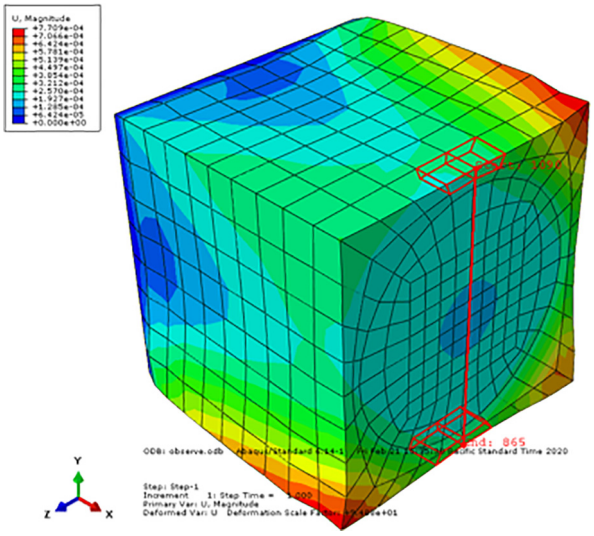


Fig. 6. The observation nodes for training.

Fig. 7. The boundary conditions of the fiber-reinforced composites plate.

Table 2
Laminate material properties and strength parameters.

	Material properties
Elastic properties	
E_1, psi	7.25×10^6
E_2, psi	1.16×10^6
G_{12}, psi	7.20×10^5
ν_{12}	0.3
Tensile strengths	
X, ksi	246.00
Y, ksi	5.10
Compressive strengths	
X', ksi	145.00
Y', ksi	17.40
Shear strengths	
S, ksi	11.60
T, ksi	6.70

$$\boldsymbol{C}_d = \frac{1}{D}\begin{bmatrix} (1-d_f)E_1 & (1-d_f)(1-d_m)\nu_{21}E_1 & 0 \\ (1-d_f)(1-d_m)\nu_{12}E_2 & (1-d_m)E_2 & 0 \\ 0 & 0 & (1-d_s)GD \end{bmatrix} \tag{15}$$

and the corresponding damaged compliance matrix is

$$\boldsymbol{H} = \begin{bmatrix} \frac{1}{(1-d_f)E_1} & -\frac{\nu_{21}}{E_2} & 0 \\ -\frac{\nu_{12}}{E_1} & \frac{1}{(1-d_m)E_2} & 0 \\ 0 & 0 & \frac{1}{(1-d_s)G_{12}} \end{bmatrix} \tag{16}$$

where $D = 1-(1-d_f)(1-d_m)\nu_{12}\nu_{21}$. The d_f, d_m, d_s reflect the current state of fiber damage, matrix damage, and shear damage respectively. As mentioned in Section 4.1, it is hard for a neural network to yield an accurate prediction of the output and the gradient of output in terms

of input at the same time. Again, the DNN model is created to map between strains and the Jacobian of $\boldsymbol{\sigma}(\boldsymbol{\varepsilon})$

$$\mathscr{M}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \sigma_{11}}{\partial \varepsilon_{11}} & \frac{\partial \sigma_{11}}{\partial \varepsilon_{22}} & \frac{\partial \sigma_{11}}{\partial \gamma_{12}} \\ \frac{\partial \sigma_{22}}{\partial \varepsilon_{11}} & \frac{\partial \sigma_{22}}{\partial \varepsilon_{22}} & \frac{\partial \sigma_{22}}{\partial \gamma_{12}} \\ \frac{\partial \sigma_{12}}{\partial \varepsilon_{11}} & \frac{\partial \sigma_{12}}{\partial \varepsilon_{22}} & \frac{\partial \sigma_{12}}{\partial \gamma_{12}} \end{bmatrix} \tag{17}$$

For this problem, $\mathscr{M}(\boldsymbol{\varepsilon},\boldsymbol{\theta})$ can be regarded as $\boldsymbol{C}_d(\boldsymbol{\varepsilon},\boldsymbol{\theta})$. Since the fiber-reinforced composite is usually assumed to be transversely isotropic, $\mathscr{M}(\boldsymbol{\varepsilon},\boldsymbol{\theta})$ can be reduced to

$$\mathscr{M}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) = C_d(\boldsymbol{\varepsilon},\boldsymbol{\theta}) = \begin{bmatrix} C_{11}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) & C_{12}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) & 0 \\ C_{12}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) & C_{22}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) & 0 \\ 0 & 0 & C_{66}(\boldsymbol{\varepsilon},\boldsymbol{\theta}) \end{bmatrix} \tag{18}$$

The DNN is fully coupled with Abaqus as $\mathscr{M}$ is a function of $\boldsymbol{\varepsilon}$. The DNN input needs to be derived from the DNN output of the previous iteration. Besides, the strains at each element might be different, which could yield different degrees of damage. Thus, the DNN outputs for each element could be different. Additionally, all the DNN outputs will contribute to the displacement. It is necessary to compute the gradient of each node's displacement with respect to all the outputs during backward propagation. Furthermore, this is a plane stress problem, which has three in-plane strains and four independent material variables. The dimensions of input and output of DNN are three and four, respectively.

The Abaqus-DNN structure for this problem is presented in Fig. 8. As shown in the figure, the input and output dimensions are three and four, which represent the dimension of the in-plane strains and the independent terms in the stiffness matrix. The main difference between the Abaqus-DNN structure shown in Fig. 5 and 8 is that each FE element in Fig. 8 corresponds to a neural network, while each material type in Fig. 5 corresponds to a neural network. The difference is because that the degree of damage for each element could be different.

The explicit form of the modified backward propagation equation is

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\theta}} = & \frac{1}{N}\sum_{i=1}^{N} i = 1^N 2(\boldsymbol{u}_i - \hat{\boldsymbol{u}}_i)\left(\sum_{i=1}^{M} \frac{\partial \boldsymbol{u}_i}{\partial \mathscr{M}_m}\frac{\partial \mathscr{M}_m}{\partial \boldsymbol{\theta}}\right) \\ = & \frac{2}{N}\left[(u_1-\hat{u}_1)\left(\frac{\partial u_1}{\partial y_{11}}\frac{\partial y_{11}}{\partial \boldsymbol{\theta}}\right) + (u_1-\hat{u}_1)\left(\frac{\partial u_1}{\partial y_{12}}\frac{\partial y_{12}}{\partial \boldsymbol{\theta}}\right) + \cdots + (u_1-\hat{u}_1)\left(\frac{\partial u_1}{\partial y_{M4}}\frac{\partial y_{M4}}{\partial \boldsymbol{\theta}}\right)\right] \\ & + \frac{2}{N}\left[(u_2-\hat{u}_2)\left(\frac{\partial u_2}{\partial y_{11}}\frac{\partial y_{11}}{\partial \boldsymbol{\theta}}\right) + (u_2-\hat{u}_2)\left(\frac{\partial u_2}{\partial y_{12}}\frac{\partial y_{12}}{\partial \boldsymbol{\theta}}\right) + \cdots + (u_2-\hat{u}_2)\left(\frac{\partial u_2}{\partial y_{M4}}\frac{\partial y_{M4}}{\partial \boldsymbol{\theta}}\right)\right] \\ & + \quad \vdots \\ & \frac{2}{N}\left[(u_N-\hat{u}_N)\left(\frac{\partial u_N}{\partial y_{11}}\frac{\partial y_{11}}{\partial \boldsymbol{\theta}}\right) + (u_N-\hat{u}_N)\left(\frac{\partial u_N}{\partial y_{12}}\frac{\partial y_{12}}{\partial \boldsymbol{\theta}}\right) + \cdots + (u_N-\hat{u}_N)\left(\frac{\partial u_N}{\partial y_{M4}}\frac{\partial y_{M4}}{\partial \boldsymbol{\theta}}\right)\right] \end{aligned} \tag{19}$$

where y_{pq} is the q_{th} neural network output of the p_{th} element.

For this problem, the FE model converged with 200 C4PSR elements. Thus, the rest of the analysis was based on this element size. Besides, this model can also be regarded as x and y axis-symmetric if the observation reference point is moved to the center of the model ($x = 1, y = 0.5$). After modifying the controlled displacement to 0.05 inch (due to symmetry), a quarter of the model was used for the analysis. The even nodes in the FE model were used for training. After trial-and-error testing, the DNN structure was identified as $[3, 40, 40, 40, 4]$, which indicates that the neural network has three inputs, four outputs, and three hidden layers with each hidden layer have 40 neurons. Again, the activation function was chosen to be the ReLU function. The training data was split into 100 datasets. This means that the training has 100 steps. The training process followed the flowchart described in Fig. 3.

4.3. Implementation of Abaqus coupled deep neural network mechacnis system

For the implementation of the Abaqus-DNN, we leveraged PyTorch to construct a neural network. PyTorch is an open-source symbolic tensor manipulation software library developed by Facebook [34]. PyTorch uses automatic differentiation in its library, which can be used for the forward and backward propagation. The PyTorch sequential function was used to build a multi-layered neural network.

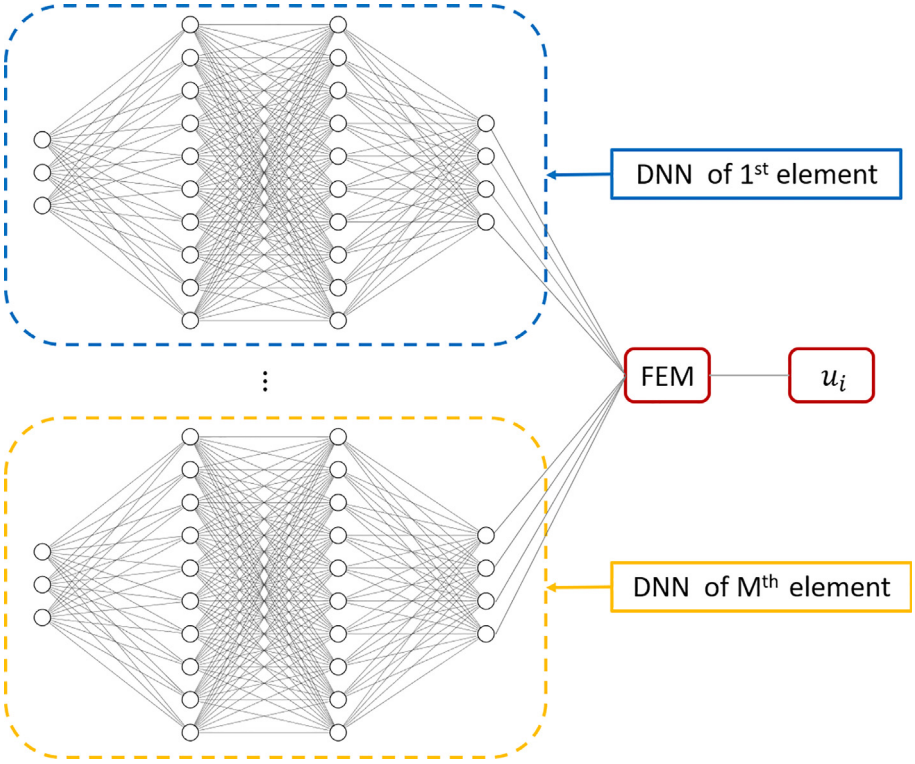


Fig. 8. The structure of Abaqus-DNN for a damage problem.

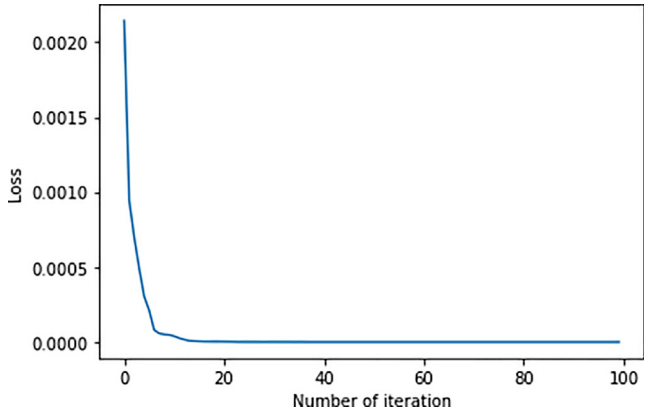


Fig. 9. Training loss of Example 1.

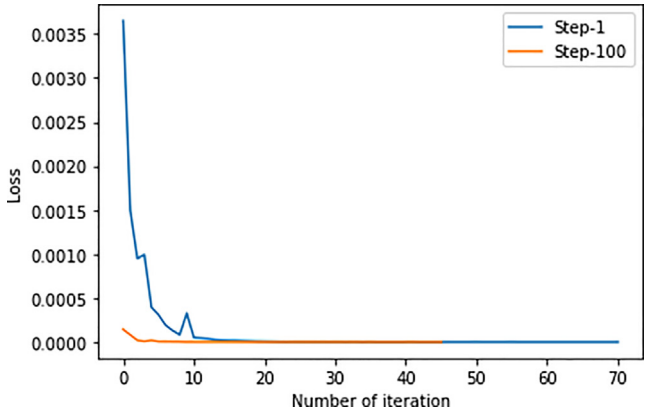


Fig. 10. Training loss of Example 10.

The FE analysis was done by Abaqus. An Abaqus python script was utilized to access the Abaqus analysis result (odb file). Unfortunately, Abaqus python was compiled based on python 2, which does not support the PyTorch package. Therefore, we have to separate the Abaqus python script and neural network optimization script. The data communication between Abaqus and neural network was achieved by saving the observed nodes' displacements, strains, and the corresponding gradients to disk using python NumPy savez function. Then, the saved file was read in another script, which will normalize the displacements to the range $[0, 1]$ and follow Eq. (13) and (19) to form the backward propagation equation.

The optimization was done through PyTorch generic L-BFGS optimizer. The training followed the process described in Fig. 3. The training would be terminated after the loss reached the accuracy or the training step reached the maximum steps. The training datasets for both problems were relatively small. Thus, a quad-core computer was used to do the training. The linear problem was completed within 20 min, the damage problem used 10 h to complete the training.

5. Results and discussions

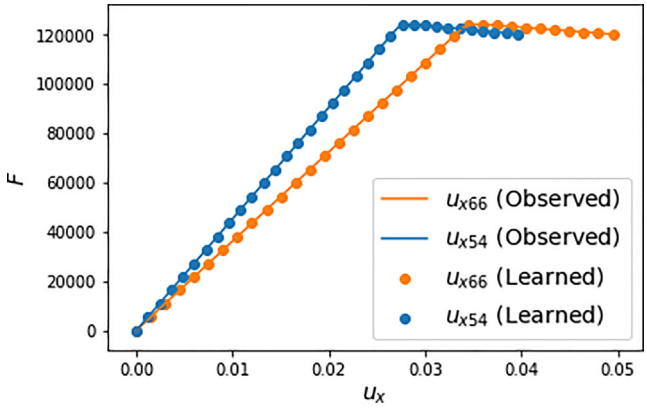
5.1. Linear elastic constitutive laws of the fiber and matrix

Fig. 9 shows that the system loss of training data with respect to the number of iterations. As can be seen in the figure, the loss decreases fast with the increase of iterations. The loss function Eq. (7) reached an optimal value of 10^{-14} within 100 iterations.

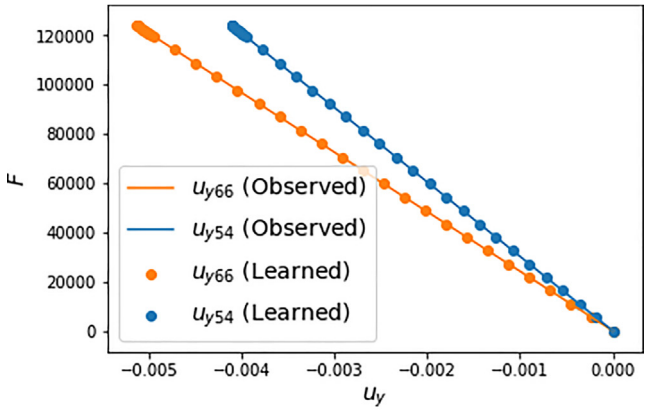
An initial guess is needed to start optimization for the L-BFGS optimizer. In principle, an ideal algorithm is desired to handle any initial guess. However, a reasonable initial guess is usually needed for an optimization problem, as a bad initial guess might cause the algorithm to converge slowly or lose convergence. For example, a negative initial guess of Young's modulus will cause the stiffness matrix no longer positive definite. Abaqus will stop running as the stiffness matrix does not physically make sense. This will cause the algorithm to stop running and lose convergence. For this problem, the initial guesses of the material properties are listed in Table 3. The initial weights and biases of the neural network model were initialized by pre-training the model based on the initial guess of the material properties. The Diff1 column shows the difference between the initial guess and the target of the material properties. As one can tell, the difference between the initial guess and the target material properties ranges from -104% to 95%. Noting that the initial guess does not have to be limited to this range, a larger range can also lead to an accurate learning result. Diff2 represents the learned material properties deviate from the target material properties. Although only several points on the exterior surface were

Table 3
Comparison between target, initial, and learned material properties.

	Target	Initial	Learned	Diff1	Diff2
E_1 (GPa)	276.00	394.25	276.01	−42.84 %	0.00 %
E_2 (GPa)	19.50	0.94	19.50	95.18 %	0.00 %
G_{12} (GPa)	70.00	50.00	70.00	28.57 %	0.00 %
v_{12}	0.28	0.20	0.28	28.57 %	0.00 %
G_{23} (GPa)	7.62	0.24	7.62	96.85 %	0.00 %
E_m (GPa)	4.76	9.73	4.76	−104.41 %	0.00 %
v_m	0.36	0.45	0.36	−25.00 %	0.00 %

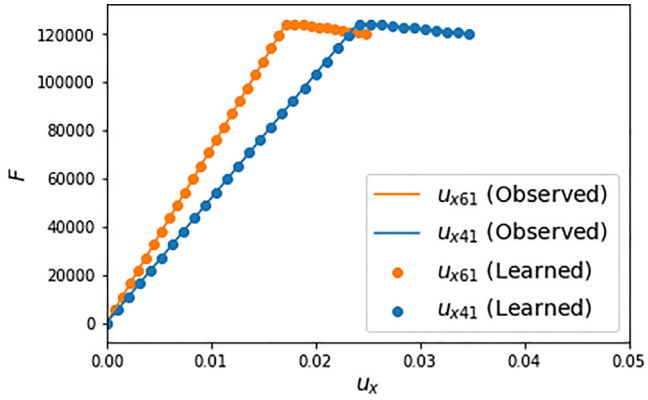


(a) Plot of F versus u_x

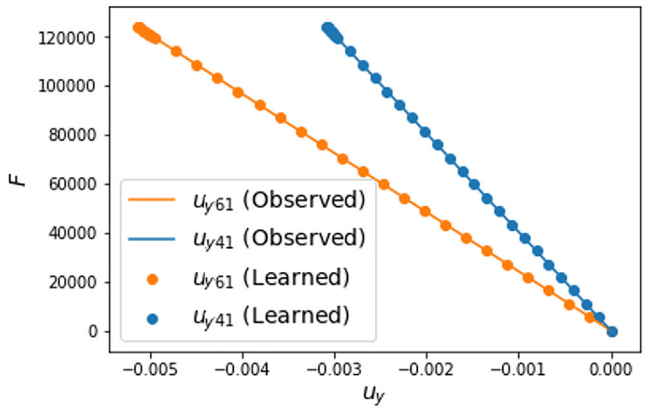


(b) Plot of F versus u_y

Fig. 11. The comparison of displacements at observed nodes.

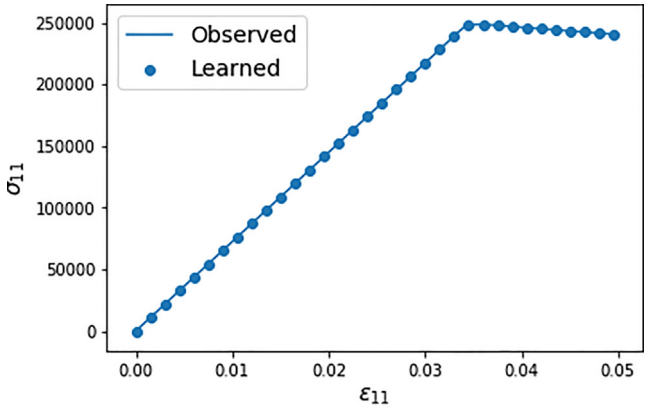


(a) Plot of F versus u_x

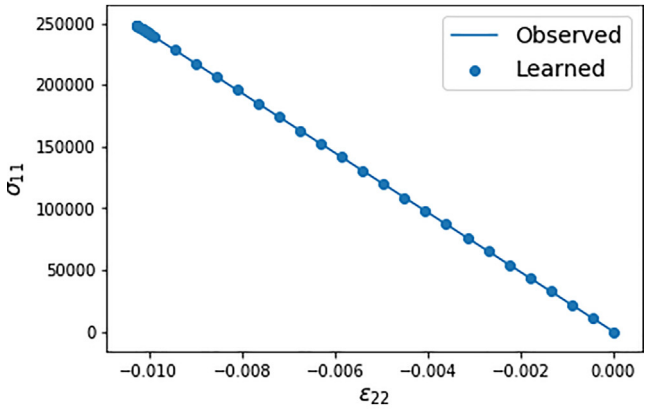


(b) Plot of F versus u_y

Fig. 12. The comparison of displacements at unobserved nodes.



(a) Plot of σ_{11} versus ϵ_{11}



(b) Plot of σ_{11} versus ϵ_{22}

Fig. 13. The comparison of stress versus strain.

selected for the training, the material properties of the two constituents can still be learned very accurately.

5.2. Damage constitutive law of a fiber-reinforced composite

Fig. 10 shows the system loss of training data to the number of iterations of step 1 and 100. These two steps are the first and last steps of the training process. The loss termination accuracy was chosen to be 10^{-12}, which is less accurate compared to the previous problem. This change was made due to the problem complexity and computational efficiency. As shown in Fig. 10, both losses decrease fast with the increase of iterations and reach the optimal value within 70 iterations.

Fig. 11 plotted the comparison of F versus u between Abaqus-DNN result and observed result at nodes 66 and 54 that were used for training. This figure shows that the learned result matches the observed result very well. This is expected as the system loss function achieved target accuracy. Besides, a comparison of the displacements at nodes 61 and 41 was presented. These nodes were not used for training. As shown in Fig. 12, the displacements predicted by the Abaqus-DNN learned model agree with the observed displacements. This indicates that it is possible to use Abaqus-DNN model for prediction based on partial observation datasets.

The constitutive law learned by the Abaqus-DNN model is presented in Fig. 13. For Fig. 13a, the initial linear curve represents the initial elasticity. Once the damage initiates, the stress–strain curve started turning. The plot in Fig. 13b shows a linear stress–strain curve, this indicates that the coupling between ε_{11} and ε_{22} remains unchanged. This result is expected, as the constitutive law to be learned assumed that the compliance matrix coupling terms are not degraded. Besides, one can also tell that the learned constitutive law matches the observed one very well. Considering only partial experimental measurements were used for training, this demonstrated that Abaqus-DNN is able to learn nonlinear constitutive law based on partially observed data.

Additionally, it is also noted that the DNN presented in this example did not fully learn the constitutive law of the material, as the training just considered one loading scenario. To fully capture the constitutive law, a series of bi-axial tests data are needed to feed into the system. This can be easily completed, as the optimization algorithm and training scheme are the same, we only need to feed the experiment datasets of different scenarios into Abaqus-DNN. Since this example has demonstrated the Abaqus-DNN mechanics system workflow and the capability of Abaqus-DNN to learn the nonlinear constitutive law, we will not perform further training of the Abaqus-DNN. But, it would be of interest to apply real experimental datasets of different loading scenarios to Abaqus-DNN to fully learn the constitutive law in future work.

6. Conclusion

In this paper, an Abaqus coupled DNN mechanics system was presented. The proposed approach enables data communication between Abaqus and DNN, which leverages the versatile FE analysis ability of Abaqus and the powerful machine learning using DNN. Abaqus-DNN makes it possible to use limited indirect measurements to learn the constitutive law in a form-free manner, which avoids the accuracy loss caused by the presumed expressions in the constitutive law. The

Abaqus-DNN was applied to learn the full set of engineering constants of the constituents of fiber-reinforced composite based on a 3D model. In addition, another example was presented to learn the nonlinear anisotropic damage constitutive law of a composite based on a plane stress model. The corresponding modified backward propagation equations were derived. The results show that the Abaqus-DNN is able to learn the constitutive laws based on structural level data, which provides a general approach to discover unknown physics by coupling neural network with commercial FE code.

Abaqus-DNN has the potential to learn other nonlinear constitutive laws, such as the time-dependent constitutive law, i.e. the viscoelastic constitutive law. For this scenario, the deformation with respect to time will be the observation data. The training would be similar to the damage problem presented in this work. Another interesting future work is to implement different techniques to improve DNN accuracy and convergence, such as adding physical-constraint to the neural network [35,36] or using symmetric positive definite neural network [20]. Finally, it would also be interesting to apply real experimental datasets of different loading scenarios to Abaqus-DNN to fully learn the constitutive law.

CRediT Author statement

Fei Tao: Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing. **Xin Liu:** Conceptualization, Software, Writing - review & editing. **Haodong Du:** Writing - review & editing, Resources. **Wenbin Yu:** Conceptualization, Supervision, Writing - review & editing, Resources.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Liu P, Zheng J. Recent developments on damage modeling and finite element analysis for composite laminates: A review. Mater Des 2010;31:3825–34.
[2] Sun C, Vaidya R. Prediction of composite properties from a representative volume element. Compos Sci Technol 1996;56:171–9.
[3] Yu W. Simplified formulation of mechanics of structure genome. AIAA J 2019;57:4201–9.
[4] Tao F, Lyu X, Liu X, Yu W. Multiscale analysis of multilayer printed circuit board using mechanics of structure genome. Mech Adv Mater Struct 2019:1–10.
[5] Kachanov LM. Rupture time under creep conditions. Int J Fract 1999;97:11–8.
[6] Chaboche J-L. Continuous damage mechanics–a tool to describe phenomena before crack initiation. Nucl Eng Des 1981;64:233–47.
[7] Krajcinovic D, Fonseka GU. The Continuous Damage Theory of Brittle Materials, Part 1: General Theory. J Appl Mech 1981;48:809–15.
[8] Lemaitre J. A Course on Damage Mechanics. Springer Science & Business Media; 2012.
[9] Gao Z, Zhang L, Yu W. A nonlocal continuum damage model for brittle fracture. Eng Fract Mech 2018;189:481–500.
[10] Degrieck J, Van Paepegem W. Fatigue damage modeling of fibre-reinforced composite materials: Review. Appl Mech Rev 2001;54:279–300.
[11] Xu R, Yang J, Yan W, Huang Q, Giunta G, Belouettar S, Zahrouni H, Zineb TB, Hu H. Data-driven multiscale finite element method: From concurrence to separation. Comput Methods Appl Mech Eng 2020;363. 112893.
[12] Liu X, Tian S, Tao F, Du H, Yu W. How machine learning can help the design and analysis of composite materials and structures?, arXiv preprint arXiv:2010.09438 (2020)..
[13] Liu X, Gasco F, Goodsell J, Yu W. Initial failure strength prediction of woven composites using a new yarn failure criterion constructed by deep learning. Compos Struct 2019;230. 111505.
[14] Ghaboussi J, Garrett Jr J, Wu X. Knowledge-based modeling of material behavior with neural networks. J Eng Mech 1991;117:132–53.
[15] Shen Y, Chandrashekhara K, Breig W, Oliver L. Finite element analysis of v-ribbed belts using neural network based hyperelastic material model. Int J Non-Linear Mech 2005;40:875–90.
[16] Wang K, Sun W. A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. Comput Methods Appl Mech Eng 2018;334:337–80.
[17] Mozaffar M, Bostanabad R, Chen W, Ehmann K, Cao J, Bessa M. Deep learning predicts path-dependent plasticity. Proc Natl Acad Sci 2019;116:26414–20.
[18] Ghaboussi J, Pecknold DA, Zhang M, Haj-Ali RM. Autoprogressive training of neural network constitutive models. Int J Numer Meth Eng 1998;42:105–26.
[19] Huang DZ, Xu K, Farhat C, Darve E. Predictive modeling with learned constitutive laws from indirect observations, arXiv preprint arXiv:1905.12530 (2019)..
[20] Xu, K, Huang DZ, Darve E. Learning constitutive relations using symmetric positive definite neural networks, arXiv preprint arXiv:2004.00265 (2020)..
[21] Liu X, Tao F, Du H, Yu W, Xu K. Learning nonlinear constitutive laws using neural network models based on indirectly measurable data. J Appl Mech 2020:1–10.
[22] Liu X, Tao F, Yu W. A neural network enhanced system for learning nonlinear constitutive relation of fiber reinforced composites, in: AIAA Scitech 2020 Forum, p. 0396..
[23] Ghannadpour S, Barvaj AK, Tornabene F. A semi-analytical investigation on geometric nonlinear and progressive damage behavior of relatively thick laminated plates under lateral pressure and end-shortening. Compos Struct 2018;194:598–610.
[24] Lapczyk I, Hurtado JA. Progressive damage modeling in fiber-reinforced materials. Compos A 2007;38:2333–41.
[25] Haykin S. Neural Networks and Learning Machines/Simon Haykin. New York: Prentice Hall; 2009.
[26] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019;378:686–707.
[27] Nielsen MA. Neural Networks and Deep Learning, volume 25. CA, USA: Determination press San Francisco; 2015.
[28] Brownlee J. A tour of machine learning algorithms. Mach Learn Mastery 2013;25.
[29] Abaqus, 6.14 documentation, Dassault Systemes Simulia Corporation (2014)..
[30] Nocedal J, Wright S. Numerical Optimization. Springer Science & Business Media; 2006.
[31] Zienkiewicz OC, Taylor RL, Nithiarasu P, Zhu J. The Finite Element Method, volume 3, McGraw-hill London, 1977..
[32] Hashin Z. Failure Criteria for Unidirectional Fiber Composites. J Appl Mech 1980;47:329–34.
[33] Camanho PP, Dávila CG. Mixed-mode decohesion finite elements for the simulation of delamination in composite materials. National Aeronautics and Space Administration, USA: Technical Report; 2002.
[34] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library, in: Advances in neural information processing systems, pp. 8026–8037..
[35] Raissi M, Perdikaris P, Karniadakis GE. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561 (2017)..
[36] Tao F, Liu X, Du H, Yu W. Physics-informed artificial neural network approach for axial compression buckling analysis of thin-walled cylinder. AIAA J 2020.