Contents

A deep energy method for finite deformation hyperelasticity

Vien Minh Nguyen-Thanh[c], Xiaoying Zhuang[c], Timon Rabczuk[a,b,*]

[a] *Division of Computational Mechanics, Ton Duc Thang University, Ho Chi Minh City, Vietnam.*
[b] *Faculty of Civil Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam.*
[c] *Institute of Continuum Mechanics, Leibniz Universität Hannover, Appelstraße 11, 30167 Hannover, Germany.*

Abstract

We present a deep energy method for finite deformation hyperelasticitiy using deep neural networks (DNNs). The method avoids entirely a discretization such as FEM. Instead, the potential energy as a loss function of the system is directly minimized. To train the DNNs, a backpropagation dealing with the gradient loss is computed and then the minimization is performed by a standard optimizer. The learning process will yield the neural network's parameters (weights and biases). Once the network is trained, a numerical solution can be obtained much faster compared to a classical approach based on finite elements for instance. The presented approach is very simple to implement and requires only a few lines of code within the open-source machine learning framework such as Tensorflow or Pytorch. Finally, we demonstrate the performance of our DNNs based solution for several benchmark problems, which shows comparable computational efficiency such as FEM solutions.

Keywords: Machine learning, Artificial Neural Networks (ANNs), Partial Differential Equations (PDEs), hyperelasticity, deep energy method

1. Introduction

Since the 1950s we have witnessed the rapid development in machine learning (ML) including an impressive range of applications in various fields, e.g. computer vision, document classification, industrial automation, speech recognition, bioinformatics, material informatics, etc. Recently, ML has attracted attention in computational engineering and mechanics. In an attempt to reduce the computational cost while guaranteeing the accuracy of solving boundary value problems (BVPs) in multiscale problem [1, 2, 3, 4, 5], model order reduction methods have been proposed in [6, 7, 8, 9, 10]. Data-driven approaches, in which the calculations are carried out directly from experiments and thus bypass 'classical' material models, have been presented in [11, 12, 13]. Artificial neural networks (ANNs) have also been applied in computational homogenization to determine the effective moduli and effective stress based on a large collection of dataset [14, 15, 16]. Nevertheless, all these approaches require discretization methods such as finite element methods which are employed to train the network.

In general, solutions of partial differential equations (PDEs) play a key role in engineering and materials science. Due to the increasing complexity of the underlying problems, commonly computational methods for approximate solutions of PDEs including the Finite Element Method (FEM) [17, 18, 19], meshfree methods [20, 21, 22] and Isogeometric Analysis (IGA) [23, 24, 25], just to mention some, are employed as most of the challenging problems cannot be solved analytically. Many of those approaches are based on the weak form. In most cases, the mathematical properties of FEM, for instance, guarantee the convergence of the approximate solution. Although the aforementioned methods have been successfully applied to solve numerous complex

*Corresponding author: timon.rabczuk@tdtu.edu.vn
Email addresses: minh.nguyen@ikm.uni-hannover.de (Vien Minh Nguyen-Thanh), zhuang@ikm.uni-hannover.de (Xiaoying Zhuang), timon.rabczuk@tdtu.edu.vn (Timon Rabczuk)

problems, several challenges remain for instance to obtain robust and efficient solutions for certain problems with high dimensionality including coupled or ill-posed problems, to name a few.

Lagaris et al [26] proposed an interesting alternative approach based on machine learning to solve PDEs in 1998, which has not been gained too much attention until recently, probably due to the lack of efficient tools such as automatic differentiation, deep machine learning algorithms and advances in GPU technology. The key idea is to minimize a loss function, i.e. the residuum in the domain and at the boundaries of the underlying PDE, at specific collocation points in order to train a neural network. Once the biases and weights of the neural network have been obtained, the solution of the problem is readily established. Based on this idea, Raissi et al. recently introduced the physics informed deep learning [27] where a deep neural network was exploited to train the model characterized by physical laws at random points on the domain and boundaries. Sirignano and Spiliopoulos [28] extended the method to problems with high dimensionality. Then, Jens Berg and Kaj Nyström [29] developed the method to solve problems with complex geometries. Recently, Hongwei Guo et al [30] applied the method to solve the BVPs in Kirchhoff plate bending possessing the C^1 continuity property. For second-order BVPs, Anitescu et at [31] developed an adaptive collocation approach to improve the accuracy of the method; they solved both forward and inverse problems. Meanwhile, Weinan and Bing Yu [32] proposed a method to approximate trial functions utilizing deep networks to solve the weak formulation of Poisson's equation and eigenvalue problems. The value of this method compared to Sirignano's and Raissi's approaches is that only the first derivatives are needed for a second-order PDE. On the downside, this approach introduces integration errors and not all governing equations can be cast in an energy-minimization framework.

In the line of work in [32], we propose a *Deep Energy Method* (DEM) for the solutions of nonlinear finite deformation hyperelasticity problems. Deep energy methods seem to be well suited for many engineering problems, where an energy functional exists. We believe the method is elegant and simple as it requires only the definition of the energy and then a 'standard' optimizer can be employed to minimize the loss function, i.e. energy functional, to obtain the biases and weights of the deep neural network. In this approach, the unconstrained solution is approximated by a process called forward propagation and then it is constrained by boundary conditions to shape the final prediction. The loss function considered in our case is the potential energy of the system. Since the loss function contains functionals, the accuracy of the solution relies on methods used to calculate the numerical integrals. A backpropagation dealing with the gradient of the loss function is computed by automatic differentiation [33]. To search for the solution, some optimizers such as Adam [34] or L-BFGS [35] which yield the solution close to the global minimum are adopted. With the optimal network parameters, the results in the context of nonlinear elastostatic problems are predicted by our trained machine at random points on the whole domain.

The outline of the paper is as follows: Section 2 gives a short introduction to large deformation hyperelasticity and governing equations. In Section 3, the neural network to approximate the primary field is explained in detail. Afterward, the DEM for hyperelasticity is presented. In Section 4, some numerical examples are sequentially presented to demonstrate the robustness and efficiency of the DEM; this is done by comparison to reference solutions or FEM solutions. Some concluding remarks are given in the last section.

2. Governing equations in Hyperelasticity

Let us consider a body made of a homogeneous, isotropic, nonlinear hyperelastic material. The body B is described by a set of particles or material points $\boldsymbol{X}$ and is bounded by the boundary ∂B in initial configuration. The solid is deformed due to the loading $\bar{\boldsymbol{t}}$ applying on it at some certain areas (shown in Figure 1). The mapping of material points from the initial configuration to the current configuration is given by: $\varphi : B \rightarrow B_t \,/\, \boldsymbol{X} \Rightarrow \varphi(\boldsymbol{X}, t) := \boldsymbol{x} = \boldsymbol{u} + \boldsymbol{X}$.

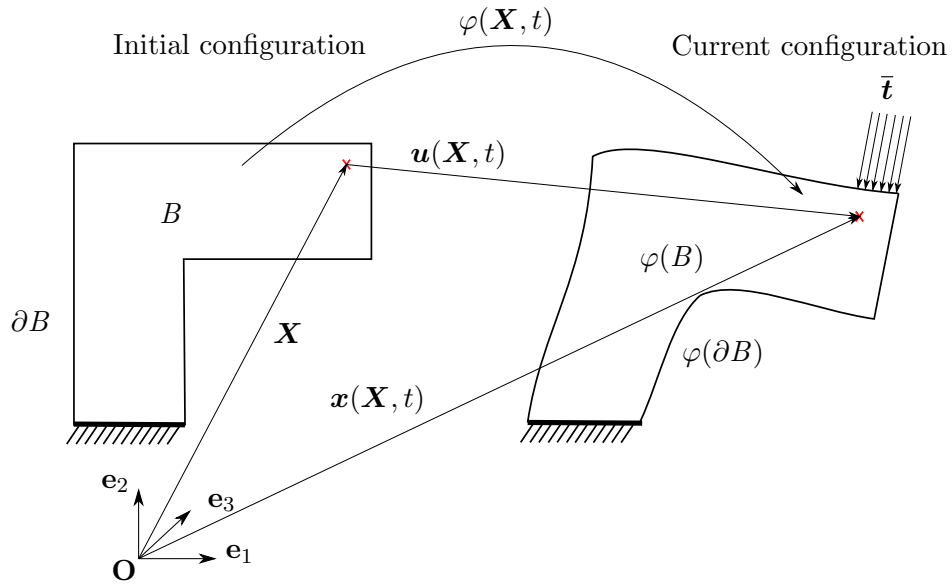


Figure 1: Motion of body B.

We opt to employ a boundary value problem in the initial configuration which consists of the following ingredients:

$$\text{Equilibrium:} \quad \nabla_X \cdot \boldsymbol{P} + \boldsymbol{f}_b = \boldsymbol{0}, \tag{1}$$

$$\text{Dirichlet boundary} : \boldsymbol{u} = \overline{\boldsymbol{u}} \quad \text{on } \partial B_u, \tag{2}$$

$$\text{Neumann boundary} : \boldsymbol{P} \cdot \boldsymbol{N} = \bar{\boldsymbol{t}} \quad \text{on } \partial B_t, \tag{3}$$

in which $\nabla_X \cdot \boldsymbol{P}$ denotes the divergence operator applying on the 1^{st} Piola-Kirchhoff stress w.r.t $\boldsymbol{X}$ on the initial configuration; $\overline{\boldsymbol{u}}$ and $\bar{\boldsymbol{t}}$ are prescribed values on Dirichlet boundary and Neumann boundary, respectively. Therein, the boundaries have to fulfill $\partial B_u \cup \partial B_t = \partial B$ and $\partial B_u \cap \partial B_t = \emptyset$. The outward normal unit vector is denoted by $\boldsymbol{N}$ and the body force is denoted by $\boldsymbol{f}_b$. The 1^{st} Piola-Kirchhoff stress $\boldsymbol{P}$ is related to its power conjugate $\boldsymbol{F}$, so-called deformation gradient, by a constitutive law. The deformation gradient is defined as

$$\boldsymbol{F} = \nabla_X \boldsymbol{\varphi}(\boldsymbol{X}), \tag{4}$$

where $\nabla_X \boldsymbol{\varphi}$ denotes the gradient operator applying on the deformation mapping w.r.t $\boldsymbol{X}$ on the initial configuration. The aforementioned constitutive law is defined as follows

$$\boldsymbol{P} = \frac{\partial \Psi}{\partial \boldsymbol{F}}, \tag{5}$$

in which Ψ is the strain energy of a specific material. In this work, we focus on two hyperelastic models: Neo-Hookean and Mooney-Rivlin. These strain energy models are given in the following:

$$\textbf{Neo-Hookean:} \qquad \Psi(I_1, J) = \frac{1}{2}\lambda\,[\log(J)]^2 - \mu \log(J) + \frac{1}{2}\mu\,(I_1 - 3), \tag{6}$$

where the first invariant is defined as $I_1 = \text{trace}(\boldsymbol{C})$ and the second invariant is defined as $J = \det(\boldsymbol{F})$. The right Cauchy-Green tensor is denoted by $\boldsymbol{C}$ and is given as $\boldsymbol{C} = \boldsymbol{F}^T \cdot \boldsymbol{F}$. Lame parameters λ and μ are given as follows

$$\lambda = \frac{E\,\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}, \tag{7}$$

where E and ν denotes Young modulus and Poisson ratio, respectively.

$$\textbf{Mooney-Rivlin:} \quad \Psi(I_1, I_2, J) = c\,(J-1)^2 - d\,\log(J) + c_1\,(I_1 - 3) + c_2\,(I_2 - 3), \tag{8}$$

where the invariants I_1, I_2, and J are defined as follows

$$I_1 = \text{trace}(\boldsymbol{C}), \quad I_2 = \frac{1}{2}\left[\text{trace}(\boldsymbol{C})^2 - \text{trace}(\boldsymbol{C}\cdot\boldsymbol{C})\right], \quad J = \sqrt{\det(\boldsymbol{C})}; \tag{9}$$

c, c_1, c_2 are material constants and d is related to c_1, c_2 by $d = 2\,(c_1 + 2\,c_2)$ which assumes that the reference configuration is stress-free. Note that while a machine learning approach can further be employed to calibrate the material parameters of these models, which is perfectly in line of the presented DEM, we assume them to be known in advance for the sake of simplicity.

To solve the balance equation with essential boundary conditions in the context of finite elements, it is necessary to perform a transformation from strong form to weak form. With the aid of the principle of virtual displacements, the weak form of linear momentum can be obtained in an initial configuration. However, there exists a strain energy function Ψ by which the elastic energy stored in the body is characterized in case of a hyperelastic material [36, 37]. The classical principle of the minimum of potential energy can be reformulated based on this strain energy in the geometrically linear theory. In the finite deformation theory, it has in general, to be considered that deformations can occur which are non-unique. Therefore, only a stationary value of potential energy can be obtained. In the case of elastostatics, the potential energy reads

$$\Pi(\boldsymbol{\varphi}) = \int_B \Psi\, dV - \int_B \boldsymbol{f}_b \cdot \boldsymbol{\varphi}\, dV - \int_{\partial B_t} \bar{\boldsymbol{t}} \cdot \boldsymbol{\varphi}\, dA. \tag{10}$$

We minimize the potential energy to find the stationary point

$$\min_{\varphi \in H} \Pi(\boldsymbol{\varphi}), \tag{11}$$

where H is the space of admissible functions (trial functions).

3. Deep Energy Method for the solutions in Hyperelasticity

3.1. Network architecture

The goal of a feedforward network is to approximate a continuous function which maps an input to the corresponding output based on the universal approximation theorem [38]. The neural network (NN) is constructed by connecting neurons to neurons between layers. Therein, the first layer (0^{th}) is the input layer, the last layer (L^{th}) is the output layer, and the layers in between are hidden layers (l^{th}). Neurons in each layer are connected to every other neuron in the next layer (see Figure 2). The numbers of neurons and layers depend on a specific application. The mapping from the input to the output can be written as $\widehat{z} : \mathbb{R}^n \to \mathbb{R}^m / \boldsymbol{X} \Rightarrow \widehat{z}(\boldsymbol{X})$, in which n defines the dimension of the network and m defines the dimension of the output. Both n and m denote the number of neurons in the input layer and output layer, respectively.

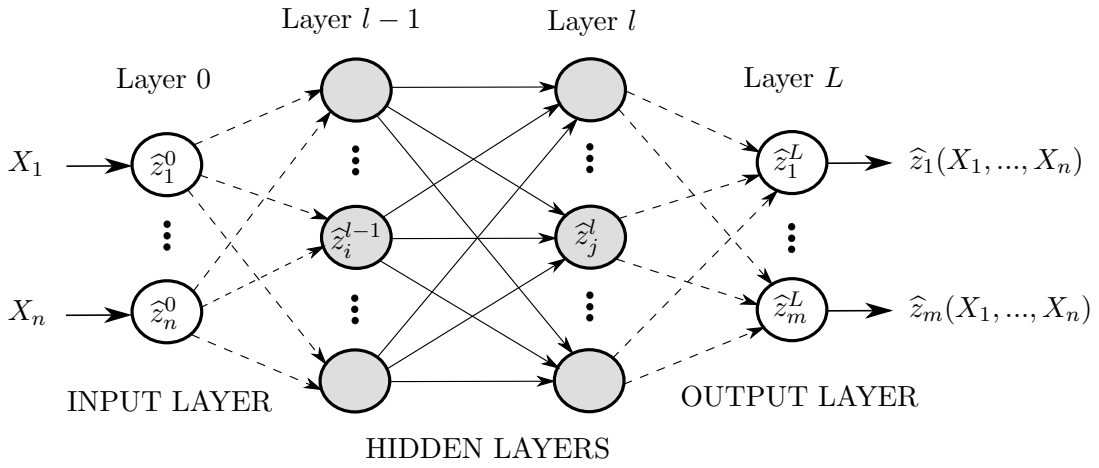


Figure 2: A feedforward neural network structure.

The activation function $\sigma : \mathbb{R} \to \mathbb{R}$ acting on a k^{th} neuron in the l^{th} layer will evaluate the value of that neuron. This is calculated based on the activation function applying to a linear combination of the neurons in the previous layer given by the corresponding weights and biases denoted by w and b, respectively. This is written as

$$\text{In the } 1^{st} \text{ layer:} \quad \widehat{z}_k^1 = \underbrace{\sigma\left(\sum_{j=1}^{n_0:=n} w_{kj}^1 X_j + b_k^1\right)}_{\sigma(z_k^1)}, \quad k = 1, ..., n_1,$$

$$\text{In the } 2^{nd} \text{ layer:} \quad \widehat{z}_k^2 = \underbrace{\sigma\left(\sum_{j=1}^{n_1} w_{kj}^2 \widehat{z}_j^1 + b_k^2\right)}_{\sigma(z_k^2)}, \quad k = 1, ..., n_2,$$

...

$$\text{In the } l^{th} \text{ layer:} \quad \widehat{z}_k^l = \underbrace{\sigma\left(\sum_{j=1}^{n_{l-1}} w_{kj}^l \widehat{z}_j^{l-1} + b_k^l\right)}_{\sigma(z_k^l)}, \quad k = 1, ..., n_l,$$

$$\text{In the last layer:} \quad \widehat{z}_k^L = \underbrace{\sigma\left(\sum_{j=1}^{n_l} w_{kj}^L \widehat{z}_j^l + b_k^L\right)}_{\sigma(z_k^L)}, \quad k = 1, ..., n_L := m.$$

Shortly, this can be written in the index notation as

$$\widehat{z}_k^l = \underbrace{\sigma\left(\sum_{j=1}^{n_{l-1}} w_{kj}^l \, \widehat{z}_j^{l-1} + b_k^l\right)}_{\sigma(z_k^l)}, \qquad 0 < l \leq L, \tag{12}$$

or in tensor form

$$\widehat{\boldsymbol{z}}^l = \underbrace{\boldsymbol{\sigma}\left(\boldsymbol{w}^l \cdot \widehat{\boldsymbol{z}}^{l-1} + \boldsymbol{b}^l\right)}_{\boldsymbol{\sigma}(\boldsymbol{z}^l)}, \qquad 0 < l \leq L, \tag{13}$$

where j is the j^{th} neuron and n_{l-1} is the number of neurons at the $(l-1)^{th}$ layer. It is necessary to find the optimal network parameters (weights and biases) so that the network's outputs are the best approximation of the actual solutions. The difference between the predicted solution and target solution is represented by an error function, called loss function, denoted by $\mathcal{L}(\widehat{\boldsymbol{z}}^L(\boldsymbol{\theta}))$ or $\mathcal{L}(\boldsymbol{\theta})$. To find these network parameters, this can be done by simply minimizing the loss function

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{14}$$

which in turn requires the gradient of the loss function with respect to the network parameters $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}; \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}}\right]^T$. In machine learning, a well-known method of computing the gradient is called backpropagation

(see Figure 3). The main steps of backpropagation are summarized as follows

1. Define the loss function: $\mathcal{L}(\widehat{\boldsymbol{z}}^L(\boldsymbol{\theta}))$.
2. Compute the gradient of $\mathcal{L}$ w.r.t $\boldsymbol{\theta}$ in the output layer: $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^L} = \left[\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}^L}; \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{b}^L}\right]^T$.

   First, we have: $\dfrac{\partial \mathcal{L}(\widehat{\boldsymbol{z}}^L(\boldsymbol{\theta}))}{\partial \boldsymbol{z}^L} = \dfrac{\partial \mathcal{L}(\widehat{\boldsymbol{z}}^L)}{\partial \widehat{\boldsymbol{z}}^L} \cdot \underbrace{\dfrac{\partial \widehat{\boldsymbol{z}}^L}{\partial \boldsymbol{z}^L}}_{\frac{\partial \boldsymbol{\sigma}(\boldsymbol{z}^L)}{\partial \boldsymbol{z}^L}} = \underbrace{\mathcal{L}' \cdot (\boldsymbol{\sigma}^L)'}_{\boldsymbol{\delta}^L}$.

   So, $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}^L} = \dfrac{\partial \mathcal{L}(\widehat{\boldsymbol{z}}^L)}{\partial \widehat{\boldsymbol{z}}^L} \cdot \dfrac{\partial \widehat{\boldsymbol{z}}^L}{\partial \boldsymbol{z}^L} \cdot \dfrac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{w}^L} = \boldsymbol{\delta}^L \cdot \widehat{\boldsymbol{z}}^l$

   and $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{b}^L} = \dfrac{\partial \mathcal{L}(\widehat{\boldsymbol{z}}^L)}{\partial \widehat{\boldsymbol{z}}^L} \cdot \dfrac{\partial \widehat{\boldsymbol{z}}^L}{\partial \boldsymbol{z}^L} \cdot \dfrac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{b}^L} = \boldsymbol{\delta}^L \cdot \mathbf{1}^L = \boldsymbol{\delta}^L$,

   where $\boldsymbol{z}^L = \boldsymbol{w}^L \cdot \widehat{\boldsymbol{z}}^l + \boldsymbol{b}^L$.
3. Compute the gradient of $\mathcal{L}$ w.r.t $\boldsymbol{\theta}$ in the l^{th} layer: $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^l} = \left[\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}^l}; \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}\right]^T$.

   Similarly, $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}^l} = \dfrac{\partial \mathcal{L}(\widehat{\boldsymbol{z}}^L)}{\partial \widehat{\boldsymbol{z}}^L} \cdot \dfrac{\partial \widehat{\boldsymbol{z}}^L}{\partial \boldsymbol{z}^L} \cdot \dfrac{\partial \boldsymbol{z}^L}{\partial \widehat{\boldsymbol{z}}^l} \cdot \dfrac{\partial \widehat{\boldsymbol{z}}^l}{\partial \boldsymbol{z}^l} \cdot \dfrac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{w}^l} = \underbrace{\boldsymbol{\delta}^L \cdot \boldsymbol{w}^L \cdot (\boldsymbol{\sigma}^l)'}_{\boldsymbol{\delta}^l} \cdot \widehat{\boldsymbol{z}}^{l-1}$

   $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \dfrac{\partial \mathcal{L}(\widehat{\boldsymbol{z}}^L)}{\partial \widehat{\boldsymbol{z}}^L} \cdot \dfrac{\partial \widehat{\boldsymbol{z}}^L}{\partial \boldsymbol{z}^L} \cdot \dfrac{\partial \boldsymbol{z}^L}{\partial \widehat{\boldsymbol{z}}^l} \cdot \dfrac{\partial \widehat{\boldsymbol{z}}^l}{\partial \boldsymbol{z}^l} \cdot \dfrac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l \cdot \mathbf{1}^l = \boldsymbol{\delta}^l$

$\Rightarrow$ In general, for the α^{th} layer where $\alpha = l, l-1, ..., 1$, we obtain:

$$\boldsymbol{\delta}^\alpha = \boldsymbol{\delta}^{\alpha+1} \cdot \ \boldsymbol{w}^{\alpha+1} \cdot (\boldsymbol{\sigma}^\alpha)', \tag{15}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^\alpha} = \boldsymbol{\delta}^\alpha \cdot \widehat{\boldsymbol{z}}^{\alpha-1}, \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^\alpha} = \boldsymbol{\delta}^\alpha. \tag{17}$$

Therein, if $\alpha = l$, these equations will become

$$\boldsymbol{\delta}^l = \boldsymbol{\delta}^L \cdot \ \boldsymbol{w}^L \cdot (\boldsymbol{\sigma}^l)',$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^l} = \boldsymbol{\delta}^l \cdot \widehat{\boldsymbol{z}}^{l-1},$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l;$$

if $\alpha = 1$, they yeild

$$\boldsymbol{\delta}^1 = \boldsymbol{\delta}^2 \cdot \ \boldsymbol{w}^2 \cdot (\boldsymbol{\sigma}^1)',$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^1} = \boldsymbol{\delta}^1 \cdot \boldsymbol{X},$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^1} = \boldsymbol{\delta}^1.$$

The learning process of an ANN then can be summarized in the following steps [1]

1. Input $[\boldsymbol{X}] = [\boldsymbol{X}_1; ...; \boldsymbol{X}_N]^T$ where N is the number of data: set the corresponding $\widehat{\boldsymbol{z}}^0([\boldsymbol{X}]) = [\boldsymbol{X}]$ for the input layer.
2. Initialize weights $\boldsymbol{w}^l$ and biases $\boldsymbol{b}^l$ for all layers.

[1] The detail discussion can be found in [39].

3. Feedforward: For each $l := 1, 2, ..., L$ compute $\boldsymbol{z}^l = \boldsymbol{w}^l \cdot \widehat{\boldsymbol{z}}^{l-1} + \boldsymbol{b}^l$ and $\widehat{\boldsymbol{z}}^l = \boldsymbol{\sigma}(\boldsymbol{z}^l)$.
4. Backpropagation:
   - Compute the gradient of loss function w.r.t the output $\boldsymbol{\delta}^L = \mathcal{L}' \cdot (\boldsymbol{\sigma}^L)'$
   - For each $\alpha := l, l-1, ..., 1$ compute $\boldsymbol{\delta}^\alpha = \boldsymbol{\delta}^{\alpha+1} \cdot \ \boldsymbol{w}^{\alpha+1} \cdot (\boldsymbol{\sigma}^\alpha)'$.
5. Gradient: The gradient of loss function is given by $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^\alpha} = \boldsymbol{\delta}^\alpha \cdot \widehat{\boldsymbol{z}}^{\alpha-1}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^\alpha} = \boldsymbol{\delta}^\alpha$.
6. Update new weights and biases based on an optimization method in every layer, e.g. in stochastic gradient descent (SGD)
   - For each $l := L, ..., 2, 1$: $\boldsymbol{w}^l_{new} = \boldsymbol{w}^l + \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^l}$ and $\boldsymbol{b}^l_{new} = \boldsymbol{b}^l + \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$ where η is the learning rate.

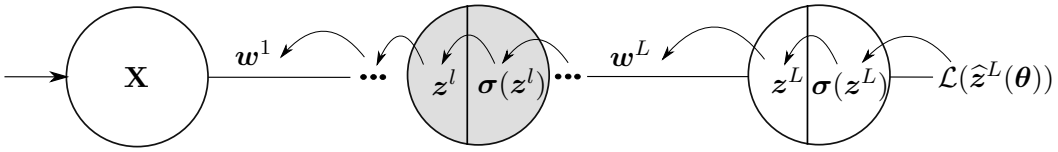


Figure 3: A schematic scheme of backpropagation.

One of the disadvantages of the proposed neural network is the nonconvexity of the loss function if the network is established by connecting neurons whose values are evaluated by nonlinear activation functions. Thus, the minimization turns into a non-convex optimization problem [40]. In most cases, we use methods in convex optimization [41] like gradient-based method (Adam optimizer), or quasi-Newton method (L-BFGS optimizer) to find the local minima.

3.2. Deep Energy Method

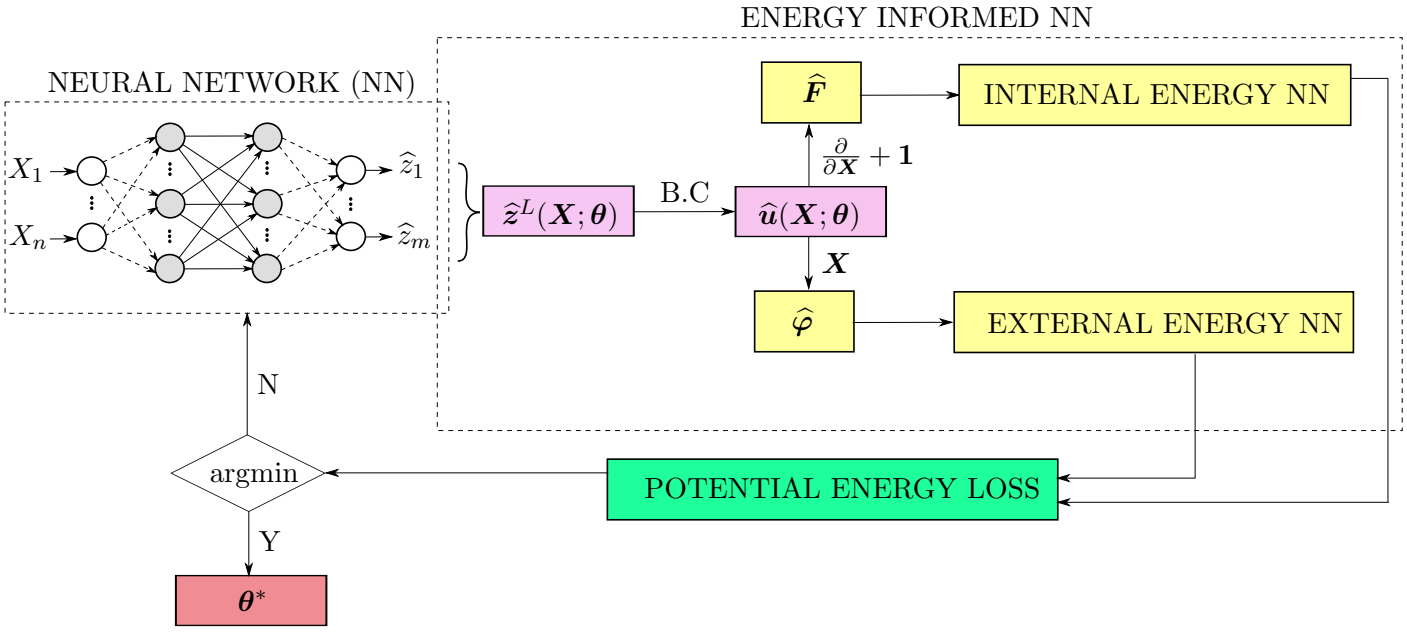


Figure 4: The whole process of a deep energy method.

The idea of the DEM is to minimize the potential energy (10) with the aid of an artificial neural network as the whole process is shown in Figure 4. We need to cast the minimization into the optimization problem

in the context of machine learning. Therefore, the definition of a loss function is essential. In this case, we exploit the potential energy as the loss function yielding

$$\underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{Potential energy loss}} = \underbrace{\int_B \Psi(\widehat{\boldsymbol{F}})\, dV}_{\text{Internal energy NN}} \underbrace{- \left(\int_B \boldsymbol{f}_b \cdot \widehat{\boldsymbol{\varphi}}(\boldsymbol{X};\boldsymbol{\theta})\, dV + \int_{\partial B_t} \overline{\boldsymbol{t}} \cdot \widehat{\boldsymbol{\varphi}}(\boldsymbol{X};\boldsymbol{\theta})\, dA \right)}_{\text{External energy NN}}, \tag{18}$$

where the hat symbols indicate the neural network's solutions. The trial function is now expressed as

$$\widehat{\boldsymbol{\varphi}}(\boldsymbol{X};\boldsymbol{\theta}) = \widehat{\boldsymbol{u}}(\boldsymbol{X};\boldsymbol{\theta}) + \boldsymbol{X}, \tag{19}$$

where $\widehat{\boldsymbol{u}}(\boldsymbol{X};\boldsymbol{\theta})$ has to fulfill boundary conditions a-priori and we use $\widehat{\boldsymbol{u}}(\boldsymbol{X})$ instead of $\widehat{\boldsymbol{u}}(\boldsymbol{X};\boldsymbol{\theta})$. A feedforward neural network constructed by network parameters $\boldsymbol{\theta}$ is employed to establish the trial solution. The displacement informed neural network is chosen in a form so that it satisfies the boundary conditions. It can be written in a form

$$\widehat{\boldsymbol{u}}(\boldsymbol{X}) = \boldsymbol{A}(\boldsymbol{X}) + \boldsymbol{B}(\boldsymbol{X}, \widehat{\boldsymbol{z}}^L(\boldsymbol{X};\boldsymbol{\theta})), \tag{20}$$

where $\widehat{\boldsymbol{z}}^L(\boldsymbol{X};\boldsymbol{\theta})$ is the output of the feedforward neural network with network parameters $\boldsymbol{\theta}$. The terms $\boldsymbol{A}(\cdot)$ and $\boldsymbol{B}(\cdot)$ are chosen in such a way that $\widehat{\boldsymbol{u}}(\boldsymbol{X})$ must satisfy the boundary conditions at certain points. By applying this procedure we can replace the original constrained optimization by an unconstrained optimization problem which is much easier to handle. The unconstrained optimization problem now is

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}), \tag{21}$$

which is solved with L-BFGS. The derivatives of the loss function $\mathcal{L}(\boldsymbol{\theta})$ with respect to the weights and biases are computed by backpropagation. Applying the mean rule, eq. (18) reads

$$\mathcal{L}(\boldsymbol{\theta}) \approx V \frac{1}{N_B} \sum_{i=1}^{N_B} \widehat{\Psi}_i - V \frac{1}{N_B} \sum_{i=1}^{N_B} (\boldsymbol{f}_b)_i \cdot \widehat{\boldsymbol{\varphi}}_i - A \frac{1}{N_{B_t}} \sum_{i=1}^{N_{B_t}} \overline{\boldsymbol{t}}_i \cdot \widehat{\boldsymbol{\varphi}}_i, \tag{22}$$

where N_B is the total number of data points in the domain; N_{B_t} is the number of data points at the traction boundary; V is the volume of the solid and A is its the surface area. If a function f is evaluated at a material point $\boldsymbol{X}_i$, we denote this by f_i. Since the mean rule results in relatively large integration error, we also tested the trapezoidal rule and Simpson's rule to evaluate the integrals, i.e.

$$\text{Trapezoidal's rule:} \quad \mathcal{L}(\boldsymbol{\theta}) \approx \sum_{i=1}^{N_B} w_i^t \widehat{\Psi}_i - \sum_{i=1}^{N_B} w_i^t [(\boldsymbol{f}_b)_i \cdot \widehat{\boldsymbol{\varphi}}_i] - \sum_{i=1}^{N_{B_t}} w_i^t [\overline{\boldsymbol{t}}_i \cdot \widehat{\boldsymbol{\varphi}}_i], \tag{23}$$

$$\text{Simpson's rule:} \quad \mathcal{L}(\boldsymbol{\theta}) \approx \sum_{i=1}^{N_B} w_i^s \widehat{\Psi}_i - \sum_{i=1}^{N_B} w_i^s [(\boldsymbol{f}_b)_i \cdot \widehat{\boldsymbol{\varphi}}_i] - \sum_{i=1}^{N_{B_t}} w_i^s [\overline{\boldsymbol{t}}_i \cdot \widehat{\boldsymbol{\varphi}}_i], \tag{24}$$

where w_i^t and w_i^s are the weights at the material point $\boldsymbol{X}_i$ of the Trapezoidal's rule and Simpson's rule, respectively, subsequently called DEMM, DEMT and DEMS where the last letters indicate the way the integral is evaluated, i.e. by mean rule, trapezoidal rule or Simpson's rule, respectively. The associated pytorch code can be found in Figure 5.

Now the elastic stored energy informed neural network has the following forms

$$\textbf{Neo-Hookean:} \quad \Psi(\widehat{I}_1, \widehat{J}) = \frac{1}{2}\lambda \left[\log(\widehat{J})\right]^2 - \mu \log(\widehat{J}) + \frac{1}{2}\mu\,(\widehat{I}_1 - 3), \tag{25}$$

$$\textbf{Mooney-Rivlin:} \quad \Psi(\widehat{I}_1, \widehat{I}_2, \widehat{J}) = c\,(\widehat{J} - 1)^2 - d\,\log(\widehat{J}) + c_1\,(\widehat{I}_1 - 3) + c_2\,(\widehat{I}_2 - 3). \tag{26}$$

where the invariants $\widehat{I}_1$, $\widehat{I}_2$, and $\widehat{J}$ are computed as follows

$$\widehat{I}_1 = \text{trace}(\widehat{\boldsymbol{C}}), \quad \widehat{I}_2 = \frac{1}{2}\left[\text{trace}(\widehat{\boldsymbol{C}})^2 - \text{trace}(\widehat{\boldsymbol{C}} \cdot \widehat{\boldsymbol{C}})\right], \quad \widehat{J} = \sqrt{\det(\widehat{\boldsymbol{C}})}. \tag{27}$$

The Green-Lagrangian strain $\widehat{\boldsymbol{E}}$ and its power conjugate 2^{nd} Piola-Kirchhoff stress $\widehat{\boldsymbol{S}}$ are given in form of NN as

$$\widehat{\boldsymbol{E}} = \frac{1}{2}\left(\widehat{\boldsymbol{C}} - \mathbf{1}\right), \quad \text{and } \widehat{\boldsymbol{C}} = \widehat{\boldsymbol{F}}^T \cdot \widehat{\boldsymbol{F}}, \tag{28}$$

$$\widehat{\boldsymbol{S}} = \widehat{\boldsymbol{F}}^{-1} \cdot \widehat{\boldsymbol{P}}, \quad \text{and } \widehat{\boldsymbol{P}} = \frac{\partial \widehat{\Psi}}{\partial \widehat{\boldsymbol{F}}}, \tag{29}$$

where $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{P}}$ denote the deformation gradient and 1^{st} Piola Kirchhoff stress tensor, respectively, informed by NNs. $\widehat{\boldsymbol{F}}$ is calculated as

$$[\widehat{F}_{iJ}] = \begin{bmatrix} \widehat{u}_{1,1} + 1 & \widehat{u}_{1,2} & \widehat{u}_{1,3} \\ \widehat{u}_{2,1} & \widehat{u}_{2,2} + 1 & \widehat{u}_{2,3} \\ \widehat{u}_{3,1} & \widehat{u}_{3,2} & \widehat{u}_{3,3} + 1 \end{bmatrix}. \tag{30}$$

Trapezoidal's rule method integrated into Pytorch's environment

```
def trapz(self, y, x=None, dx=1.0, axis=-1):
    if x is None:
        d = dx
    else:
        d = x[1:] - x[0:-1]
        # reshape to correct shape
        shape = [1] * y.ndimension()
        shape[axis] = d.shape[0]
        d = d.reshape(shape)
    nd = y.ndimension()
    slice1 = [slice(None)] * nd
    slice2 = [slice(None)] * nd
    slice1[axis] = slice(1, None)
    slice2[axis] = slice(None, -1)
    ret = torch.sum(d * (y[tuple(slice1)] + y[tuple(slice2)]) / 2.0, axis)
    return ret
```

Simpson's rule method integrated into Pytorch's environment

```
def _basic_simps(self, y, start, stop, x, dx, axis):
    nd = len(y.shape)
    if start is None:
        start = 0
    step = 2
    slice_all = (slice(None),) * nd
    slice0 = self.tupleset(slice_all, axis, slice(start, stop, step))
    slice1 = self.tupleset(slice_all, axis, slice(start + 1, stop + 1, step))
    slice2 = self.tupleset(slice_all, axis, slice(start + 2, stop + 2, step))

    if x is None:  # Even spaced Simpson's rule.
        result = torch.sum(dx / 3.0 * (y[slice0] + 4 * y[slice1] + y[slice2]), axis)
    else:
        # Account for possibly different spacings.
        #    Simpson's rule changes a bit.
        h = self.torch_diff_axis_0(x, axis=axis)
        sl0 = self.tupleset(slice_all, axis, slice(start, stop, step))
        sl1 = self.tupleset(slice_all, axis, slice(start + 1, stop + 1, step))
        h0 = h[sl0]
        h1 = h[sl1]
        hsum = h0 + h1
        hprod = h0 * h1
        h0divh1 = h0 / h1
        tmp = hsum / 6.0 * (y[slice0] * (2 - 1.0 / h0divh1) +
                            y[slice1] * hsum * hsum / hprod +
                            y[slice2] * (2 - h0divh1))
        result = torch.sum(tmp, dim=axis)
    return result
```

Figure 5: Trapezoidal's rule and Simpson's rule are programed in Pytorch's environment using Pytorch operators based on the integration module in NumPy and SciPy. It is not much difficult to adapt from NumPy/SciPy functions/operators to Pytorch functions/operators due to the same syntax. The blue circles indicate the Pytorch tensors. The red circles indicate the Pytorch operators. The green rectangles indicate the Pytorch methods.

3.3. Implementation

The DEM is implemented based on object-oriented programming (OOP) reusing and inheriting the avail-

able objects (Figure 6). Here, the main class is **DeepEnergyMethod** which invokes the public methods: *trainModel()* and *evaluateModel()*. The former method will train the defined model structure by feeding the data points and calculate/update the new weights, biases by optimizing the approximate potential loss to search for local minima (Algorithm 2). The latter method returns the displacement, the deformation gradient, Green-Lagrangian strain and second Piola-Kirchhoff stress based on the trained knowledge of the machine. The DEM procedure is summarized in Algorithm 1.
The deep neural networks are defined in a separate class called **MultiLayerNet** where the *forward()* and *backward()* indicate the forward propagation and backward propagation presented in 3.1. Different types of numerical integration based on Pytorch's environment and Pytorch's operators are implemented in the **IntegrationLoss** class. The material models or the elastic energy is integrated with the **EnergyModel** class. Since the optimization algorithm is not the focus of our work, we utilize the available build-in optimizers defined in Pytorch framework.

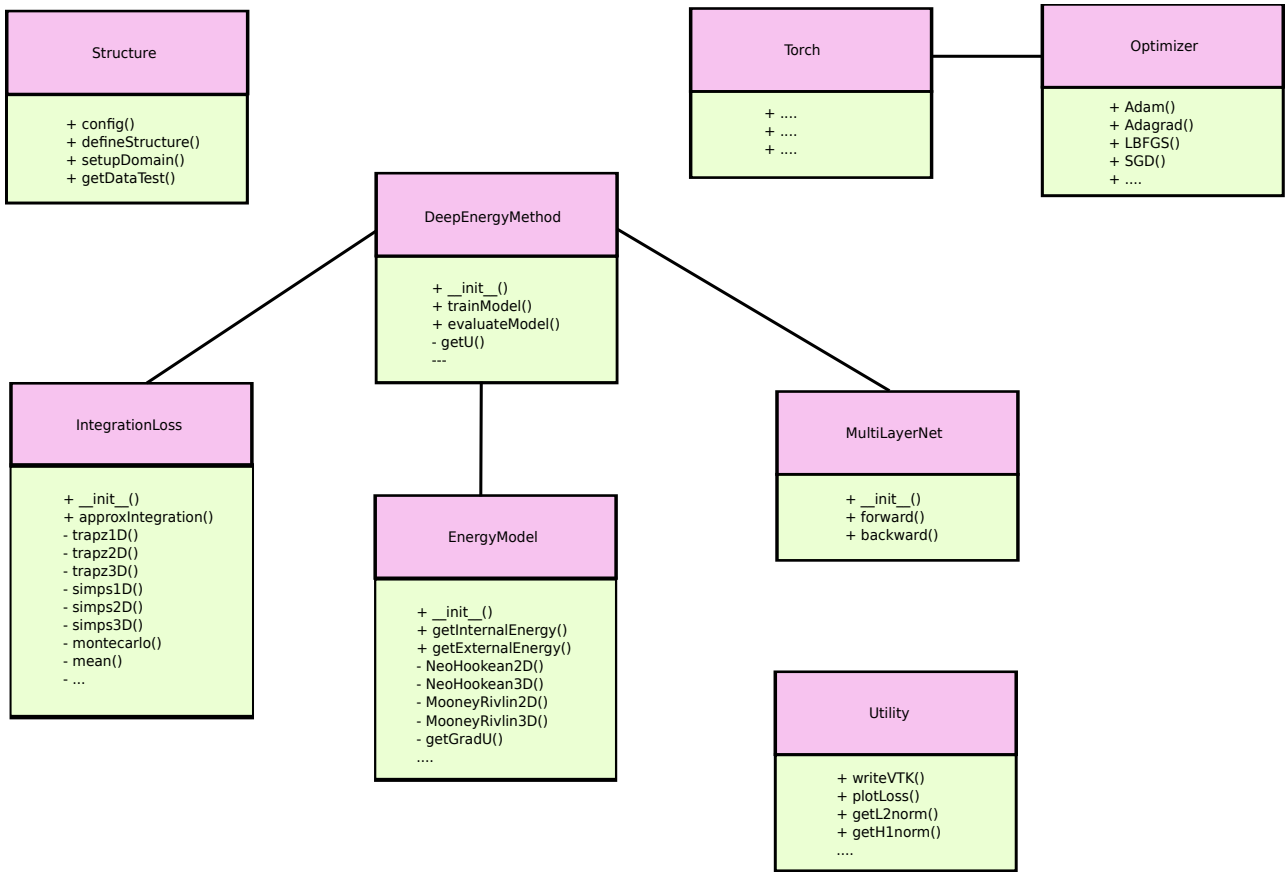


Figure 6: The schematic UML (Unified Modeling Language) for Deep Energy Method. The program is designed based on the concept of OOP. One class must contain one or several attributes (not shown in figure for the sake of simplicity) and one or several methods ('+' sign denotes the public method while '-' sign denotes the private methods).

ALGORITHM 1: Main algorithm of Deep Energy Method.

Input : dim – dimensional problem
eType – type of energy model
matParams – material parameters
nnStruct – neural network structure
netParams – network parameters such as iteration number, learning rate, ...
intType – type of numerical integration
X – coordinates ($\boldsymbol{X} = [X_1, X_2, X_3]^T$) of all data
Xf – coordinates of data subjected to loading
Xu – coordinates of data subjected to displacement constrain
XTest – coordinates of data which will be predicted

Output: U – displacement of the predicted data
F – deformation gradient of the predicted data
E – Green-Lagrangian strain of the predicted data
S – Second Piola-Kirchhoff stress of the predicted data

```
// initialize an instance of the material class
mat ← EnergyModel(eType, matParams)
// initialize an instance of the deep energy method class
dem ← DeepEnergyMethod(nnStruct, intType, mat, dim)
// train our model
dem.trainModel(X, Xf, Xu, netParams)
// evaluate our model
U, F, E, S ← dem.evaluateModel(XTest)
```

ALGORITHM 2: Training algorithm of Deep Energy Method class.

Input : netParams – network parameters
X – coordinates ($\boldsymbol{X} = [X_1, X_2, X_3]^T$) of all data
Xf – coordinates of data subjected to loading
Xu – coordinates of data subjected to displacement constrain

Output: w – neural network's weights
b – neural network's biases

```
/* Initialize an optimizer instance.  Here, we use L-BFGS optimization method for our solution.  */
optimizer ← torch.optim.LBFGS(self.model.parameters(), netParams)
/* In each training step, we will calculate and update network parameters.                       */
for i in netParams.iter do
    /* Constrain the displacement so that it satisfies the boundary conditions                   */
    u ← self.getU(X, Xu)
    /* Calculate the potential field                                                             */
    φ ← u + X
    /* Get the internal-energy integrand informed from our neural network                        */
    eInt ← self.energy.getInternalEnergy(u, X)
    /* Get the external-energy integrand informed from our neural network                        */
    eExt ← self.energy.getExternalEnergy(φ, Xf)
    /* Approximate the internal energy                                                           */
    eIntApprox ← self.intLoss.approxIntegration(eInt)
    /* Approximate the external energy                                                           */
    eExtApprox ← self.intLoss.approxIntegration(eExt)
    /* Establish the final form of loss function based on the balance of energy principle        */
    energyLoss ← eIntApprox + eExtApprox
    /* Execute the backpropagation which takes the derivative of energy loss w.r.t weights and biases */
    energyLoss.backward()
    /* Update new weights and new biases based on the L-BFGS optimizer                           */
    optimizer.step()
end
```

Finally, we summarize some key features of the proposed DEM compared to previous approaches:

- The minimization of the loss function is similar to the classical principle of minimum potential energy of finding the stationary points.

- The deformations $\widehat{\varphi}$ ensure the potential NN $\mathcal{L}$ fulfills the equilibrium state when the potential energy is minimized.
- It requires only the first gradients of the primary fields in contrast to previous approaches [42, 43, 27] which need the 2^{nd} order gradients for second order PDEs.
- Traction free boundary conditions are automatically fulfilled.
- The constrained optimization is replaced by an unconstrained optimization problem.

4. Numerical examples

In this section, we present some benchmark problems to examine the robustness and effectiveness of the DEM. For this purpose, we will compare the DEM solution with reference (analytical or FEM) solutions.

4.1. 1D problem

We first study a simple 1D bar problem as similarly depicted in [16]. The problem is described in Figure 7 and the potential energy is given as

$$\Psi(\epsilon) = (1+\epsilon)^{\frac{3}{2}} - \frac{3}{2}\epsilon - 1. \tag{31}$$

Note that $\Psi(0) = 0$ when there is no deformation. A potential energy form then becomes

$$\Pi = \int_L \Psi(\epsilon)\, dX - \int_L f(X)\, u(X)\, dX - [\bar{t}\, u(X)]_{X:=L}. \tag{32}$$

In this formulation, X are material coordinates, $\epsilon = du/dX$ denotes the displacement gradient and $\bar{t}$ is the traction force applied to the right side of the bar. We set $\bar{t}$ to be zero. This boundary value problem in the strong form reads

$$\frac{d}{dX}\frac{\partial \Psi}{\partial \epsilon} + f(X) = 0, \quad -1 \leq X \leq L := 1 \tag{33}$$

with boundary conditions

$$u(-1) = 0, \quad \frac{\partial \Psi\,(\epsilon)}{\partial \epsilon}|_{X:=L} = \bar{t}. \tag{34}$$

The exact solution in this BVP is given

$$u(X) = \frac{1}{135}(68 + 105X - 40X^3 + 3X^5). \tag{35}$$

The L^2 error norm and H^1 error seminorm are calculated as follows

$$\begin{aligned} ||e||_{L^2} &= \frac{||\widehat{u} - u||_{L^2}}{||u||_{L^2}} = \frac{\sqrt{\int_L (\widehat{u} - u)^2\, dX}}{\sqrt{\int_L u^2\, dX}}, \\ ||e||_{H^1} &= \frac{||\widehat{u} - u||_{H^1}}{||u||_{H^1}} = \frac{\sqrt{\int_L (\widehat{\epsilon} - \epsilon)^2\, dX}}{\sqrt{\int_L \epsilon^2\, dX}}. \end{aligned} \tag{36}$$

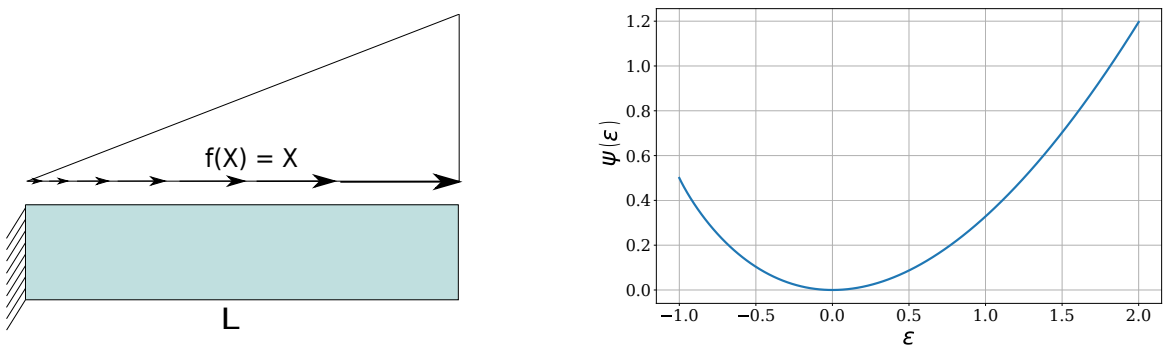
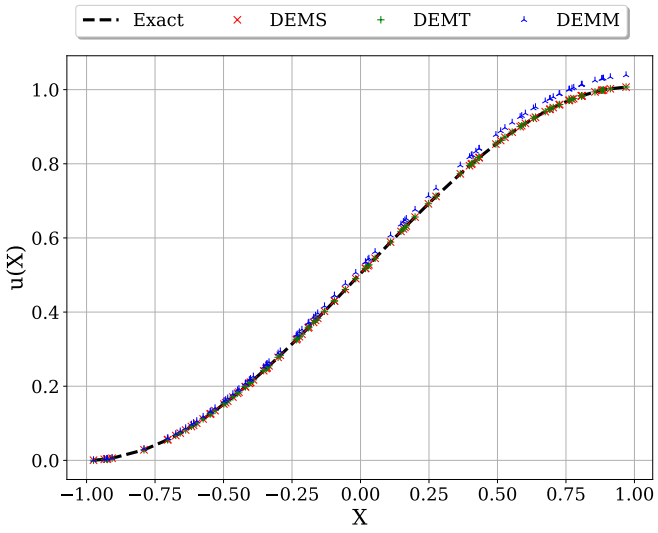


Figure 7: *Mathematical problem.* (*left*) Problem setting of the mathematical bar. (*right*) With the energy function $\Psi(\epsilon) = (1+\epsilon)^{\frac{3}{2}} - \frac{3}{2}\epsilon - 1$ is plotted against the ϵ-coordinate.

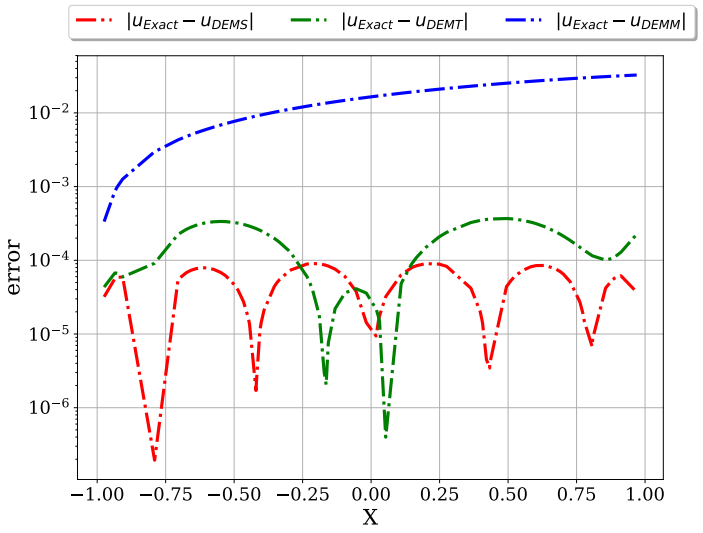
Setup We use 2 datasets: 100 points and 1000 points as the feeding data to train our network. A shallow network $(1-10-1)$ is constructed. Therein, we use 1 neuron in the input layer for the nodal coordinates and 1 neuron in the output layer for the unconstrained solution. 10 neurons are used to enforce the learning in the hidden layer. We use *tanh* activation function to evaluate neural values in the hidden layers. Besides, L-BFGS optimizer is used in this example. The neural network is strained for 30 steps.
Based on the boundary condition, the constrained NN solution has the form as

$$\widehat{u}(X) = (X+1)\,\widehat{z}^{L}(X;w,b). \tag{37}$$

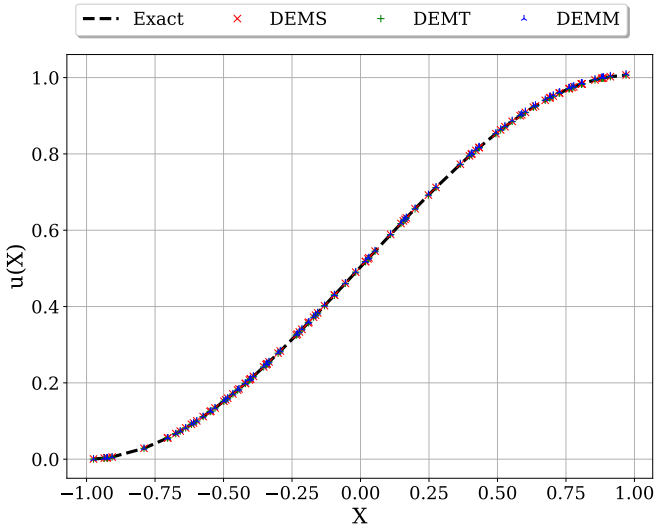
Result Figure 8 shows the displacement solution obtained by DEM for the two datasets. While the displacement solution of the DEMT and DEMS agree well with the analytical solution, the DEMM's solution is less accurate, especially at the end of the bar. This is probably due to numerical integration errors. Similar results are obtained for the displacement gradient in Figure 9.
We also compute the error in the L^2 norm and H^1 seminorm, see Table 1. DEM based on Simpson's rule always provides the best approximation in all cases.



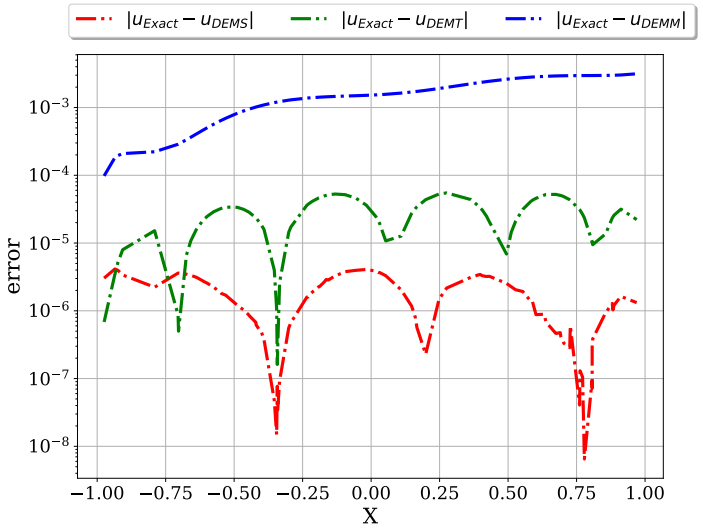
(a) Predicted solution (100 training data)



(b) The error (100 training data)

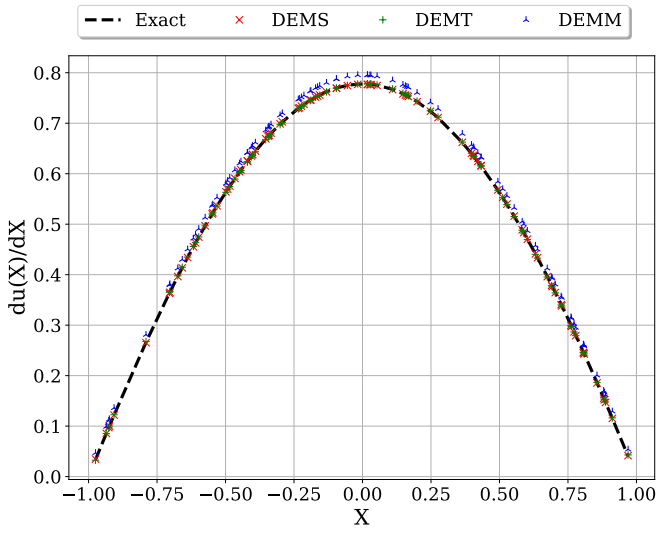


(c) Predicted solution (1000 training data)

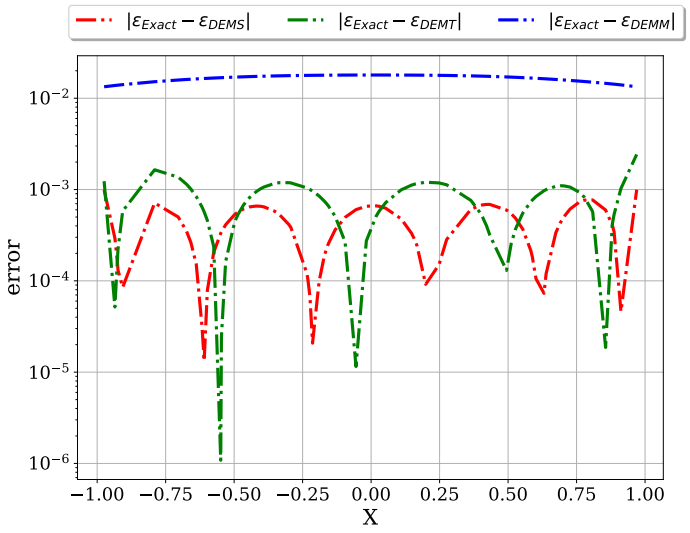


(d) Error (1000 training data)

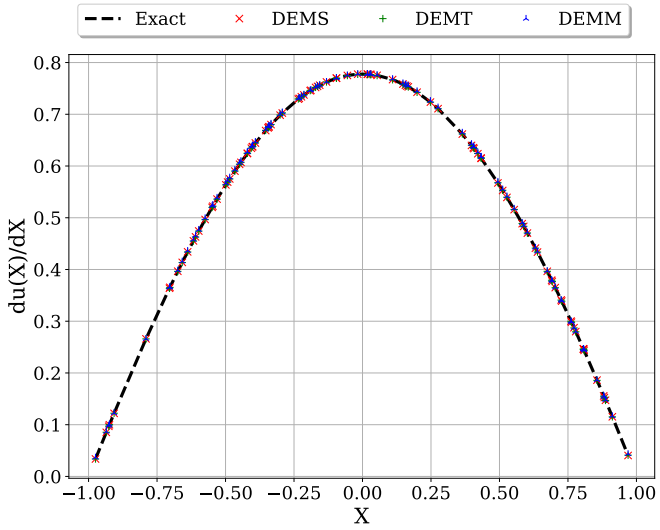
Figure 8: The displacement solution obtained by DEMM, DEMT, DEMS in two cases.



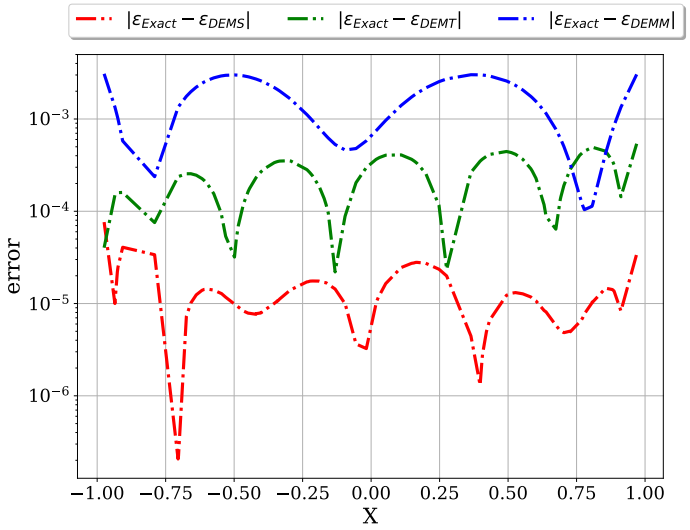
(a) Predicted du/dX (100 training data)



(b) Error (100 training data)



(c) Predicted du/dX (1000 training data)



(d) Error (1000 training data)

Figure 9: The displacement gradient solution obtained by DEMM, DEMT, DEMS in two cases.

Data	**Method**	**DEMM**	**DEMT**	**DEMS**
10^1 points (lr=0.01)	$\|\|e\|\|_{L^2}$	3.70e-01	2.66e-02	4.98e-03
	$\|\|e\|\|_{H^1}$	3.53e-01	4.04e-02	3.48e-02
10^2 points (lr=0.1)	$\|\|e\|\|_{L^2}$	3.13e-02	3.55e-04	9.39e-05
	$\|\|e\|\|_{H^1}$	2.92e-02	1.53e-03	8.95e-04
10^3 points (lr=1.0)	$\|\|e\|\|_{L^2}$	3.12e-03	5.06e-05	3.56e-06
	$\|\|e\|\|_{H^1}$	3.33e-03	5.15e-04	2.87e-05

Table 1: L^2 error norm and H^1 error seminorm comparison in 3 cases: 10 points, 100 points, 1000 points.

4.2. 2D bending beam

Next, we consider a two-dimensional bending beam problem. Plane strain conditions are assumed. The beam has the following dimensions: length $L = 4$ m, height $H = 1$ m, and depth $D = 1$ m. The beam is subjected to a uniform traction load $\bar{t} = -5.0$ N at the right end and is clamped at the left side as shown in Figure 10. The beam is made of a homogeneous, isotropic, Neo-Hookean material (6) with Young's modulus $E = 1000$ N/m^2 and Poisson's ratio $\nu = 0.3$. In 2D, the strain energy density is given as

$$\Psi(I_1, J) = \frac{1}{2}\lambda[\log(J)]^2 - \mu \log(J) + \frac{1}{2}\mu(I_1 - 2). \tag{38}$$

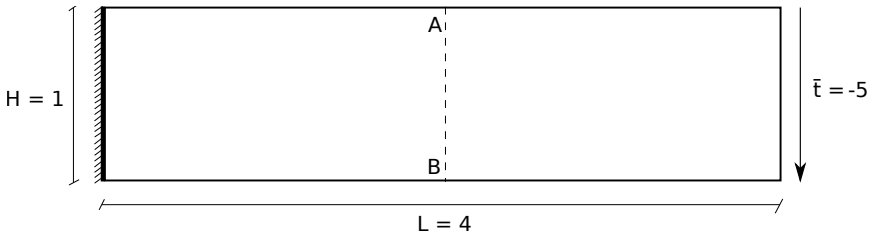


Figure 10: 2D beam bending test.

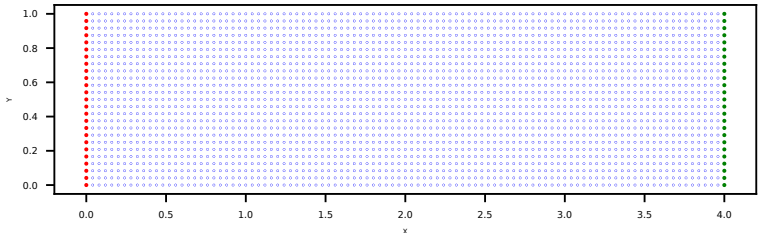


Figure 11: The training point distribution of a 2D beam bending for the training step. The red points are used to enforce/train the Dirichlet boundary condition. The green points are used to enforce/train the external load. The blue points are used for the interior domain integration.

Setup We use 2500 ($N_1 = 100$, $N_2 = 25$) points as the feeding data to train our network as shown in Figure 11. A 4-layer network ($2 - 30 - 30 - 2$) is constructed. Therein, we use 2 neurons in the input layer for the coordinates and 2 neurons in the output layer for the unconstrained solution. 30 neurons are used to enforce the learning in each hidden layer. We use *tanh* activation function to evaluate neural values in the hidden layers. Also, L-BFGS optimizer with a learning rate of 0.5 is used in this example. The neural network is strained for 20 steps.
Based on the boundary condition, the constrained NN solution has the form as

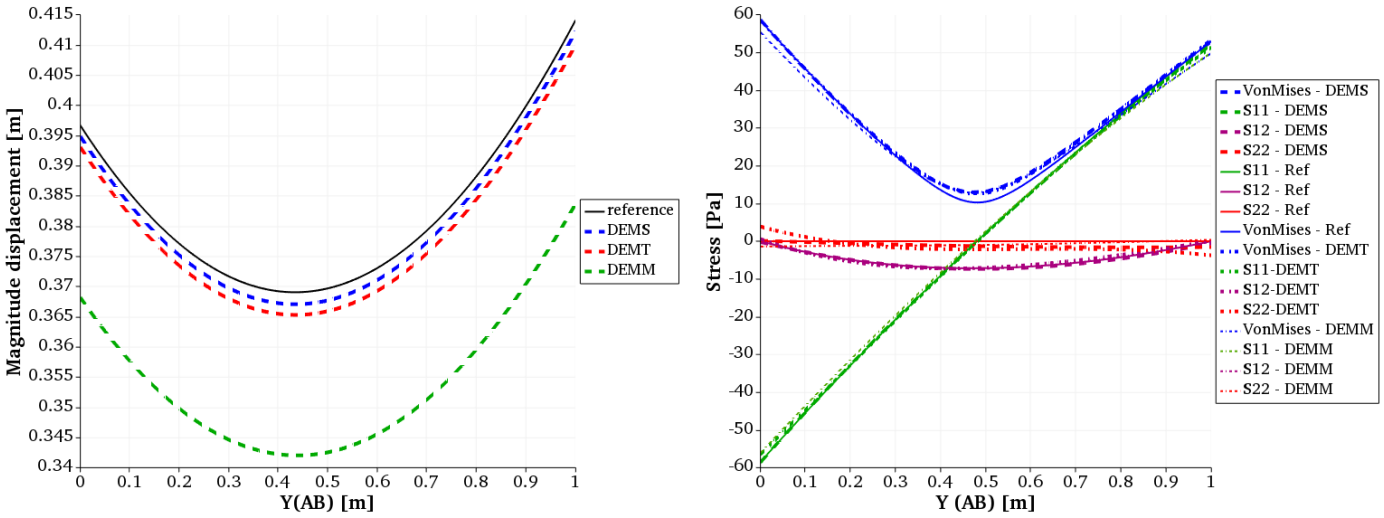
$$\begin{aligned} \widehat{u}_1(X_1, X_2) &= X_1\, \widehat{z}_1^L(X_1, X_2; w, b), \\ \widehat{u}_2(X_1, X_2) &= X_1\, \widehat{z}_2^L(X_1, X_2; w, b). \end{aligned} \tag{39}$$

Result We use FEM to simulate the beam with a fine mesh to obtain the reference solution. We first measure the displacement and stress at the AB line depicted in Figure 10. Figure 12 shows the results obtained by DEMx (all versions of DEM) compared with the reference solution. Figure 12a shows that DEMS and DEMT yield more accurate results than DEMM. The displacement trends of all DEMx agree with the reference solution. Figure 12b presents the stress results obtained by DEM compared to the reference result. Figure 13 and Figure 14 show the contour plot of the displacement in y direction and the VonMises stress.

We also verify the accuracy of the proposed method by calculating the error in L^2 norm and H^1 seminorm. Note that in this case the analytical solution is not given, so the 'error' in the L^2 norm and H^1 seminorm are calculated as follows

$$e_{L^2} = \frac{||\widehat{\boldsymbol{u}}||_{L^2} - ||\boldsymbol{u}_{Ref}||_{L^2}}{||\boldsymbol{u}_{Ref}||_{L^2}},$$
$$e_{H^1} = \frac{||\widehat{\boldsymbol{u}}||_{H^1} - ||\boldsymbol{u}_{Ref}||_{H^1}}{||\boldsymbol{u}_{Ref}||_{H^1}}. \tag{40}$$

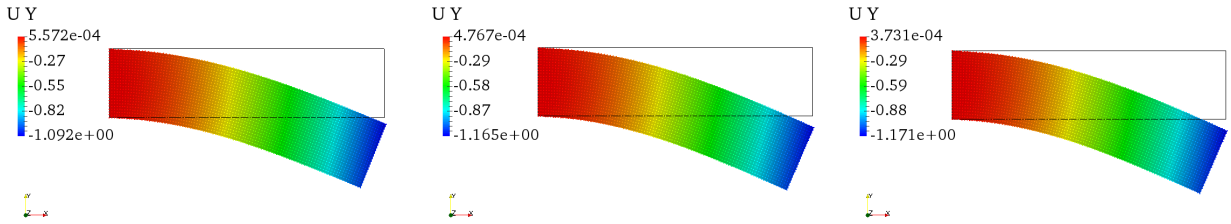
Here, four different neural networks are constructed. Each network has a different number of the hidden layers (HL), e.g. 1HL, 2HL, 3HL, 4HL. Each hidden layer contains 30 neurons to enhance the learning process. In addition, we train the NNs for 20, 40, 60, 80 steps. As we can see from Figure 15, DEMS and DEMT yield better results than DEMM. More importantly, the solution converges with respect to the training steps in both L^2 norm and H^1 seminorm . However, it does not converge with respect to the number of hidden layers. More hidden layers do not always yield better approximation.



(a) Displacement magnitude comparison

(b) Stress comparison

Figure 12: The displacement magnitude and stress comparison between DEM and the reference solution at AB in 2D bending beam test (Neo-Hookean model).

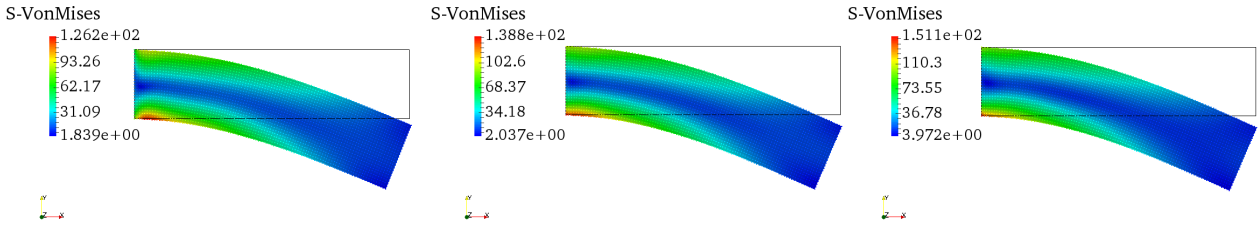


(a) u_2 - DEMM

(b) u_2 - DEMT

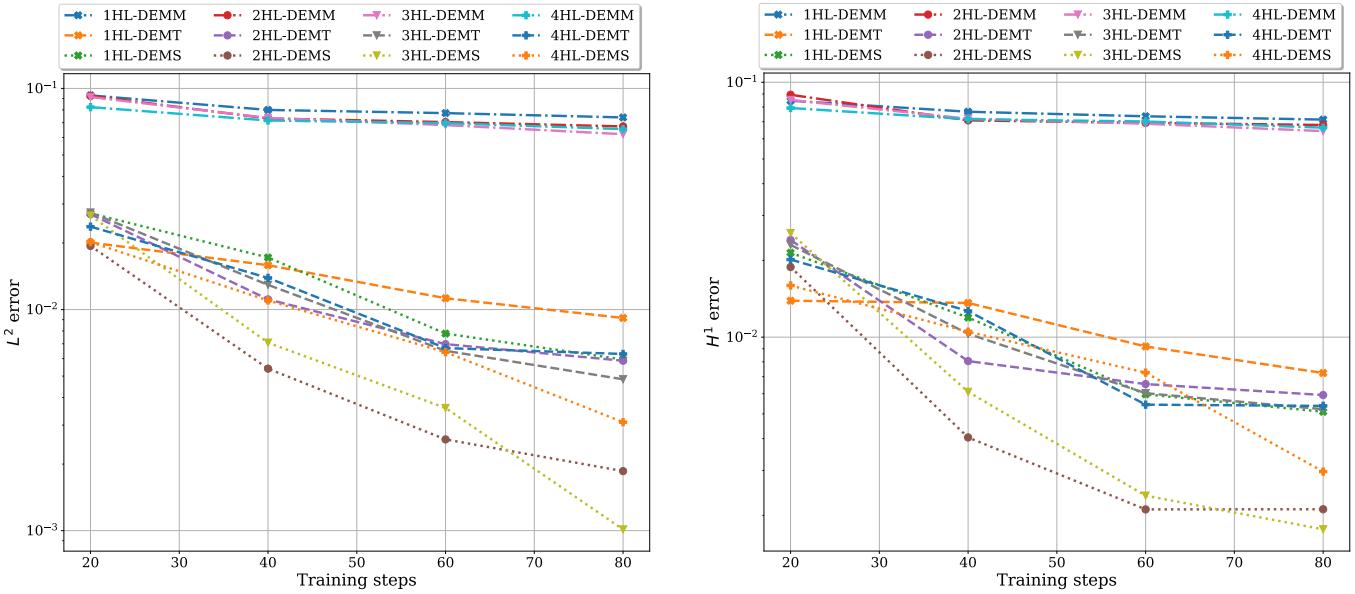
(c) u_2 - DEMS

Figure 13: The vertical displacements of all versions of DEM.



(a) VonMises stress - DEMM (b) VonMises stress - DEMT (c) VonMises stress - DEMS

Figure 14: The VonMises stresses of all versions of DEM.



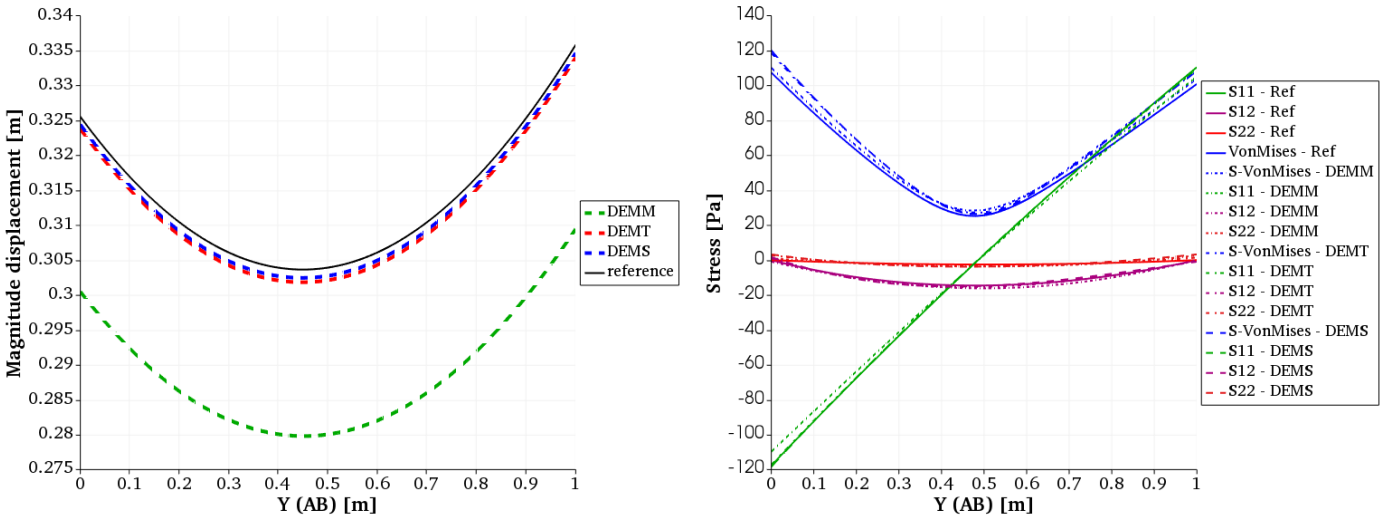
(a) L^2 norm error (b) H^1 seminorm error

Figure 15: The error in terms of L^2 norm and H^1 seminorm of DEMx with respect to the training steps.

Let's consider the Mooney-Rivlin model in this example. The strain energy density is given as

$$\Psi(I_1, I_2, J) = c(J-1)^2 - d\log(J) + c_1(I_1 - 2) + c_2(I_2 - 1), \tag{41}$$

where the material parameters are given as $c = 100$ Pa, $c_1 = 630$ Pa, $c_2 = -1.2$ Pa, and $d = 2(c_1 + 2c_2)$. The loading in this case is increased to $\bar{t} = -10$ N. Figure 16 presents the results obtained by DEMx compared to reference solution.



(a) Displacement magnitude comparison

(b) Stress comparison

Figure 16: The displacement magnitude and stress comparison between DEM and the reference solution at AB in 2D bending beam test (Mooney-Rivlin model).

4.3. 3D bending beam

In this example, we model the bending beam problem in 3D. The parameters are similar to the 2D Neo-Hookean beam bending example except for the potential energy characterized by the Neo-Hookean model which is given in 3D by

$$\Psi(I_1, J) = \frac{1}{2}\lambda[\log(J)]^2 - \mu \log(J) + \frac{1}{2}\mu(I_1 - 3). \tag{42}$$

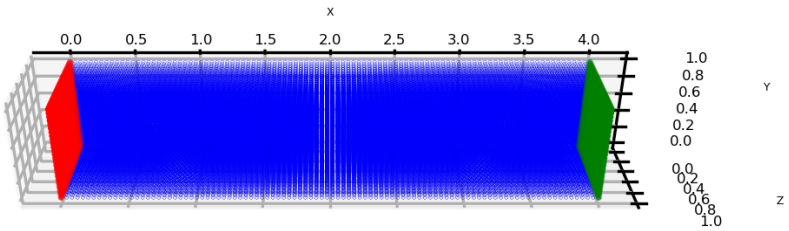


Figure 17: The training point distribution of a 3D beam bending for the training in DEM. The red points are used to enforce/train the Dirichlet boundary condition. The green points are used to train the external load. The blue points are used for the interior domain integration.

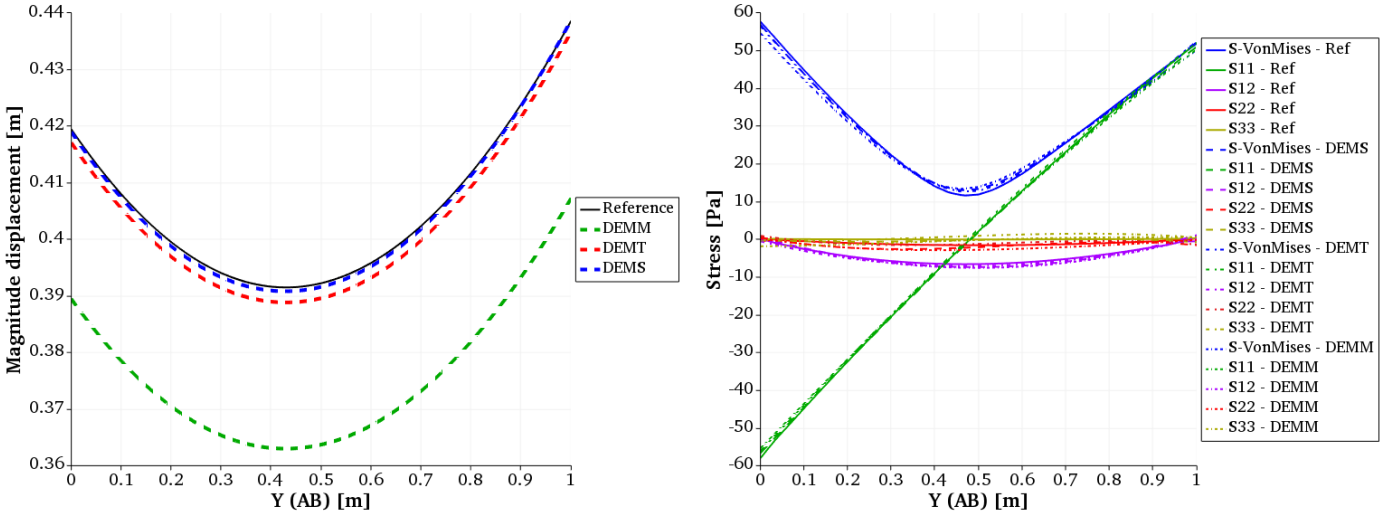
Setup We generate 62500 points ($N_1 = 100, N_2 = 25, N_3 = 25$) as the feeding data to train our network as shown in Figure 17. A 5-layer network ($3 - 30 - 30 - 30 - 3$) is constructed. Therein, we use 3 neurons in the input layer for the data coordinates and 3 neurons in the output layer for the unconstrained displacement. Moreover, in each hidden layer 30 neurons are used to enforce the learning. We use *tanh* activation function to evaluate neural values in the hidden layers. The L-BFGS optimizer with learning rate $lr = 0.5$ is employed to search for the network parameters. The neural network is strained for 40 steps. With the prescribed boundary

conditions, the components of the displacement have the form

$$\widehat{u}_1(X_1, X_2, X_3) = X_1\, \widehat{z}_1^L(X_1, X_2, X_3; w, b),$$
$$\widehat{u}_2(X_1, X_2, X_3) = X_1\, \widehat{z}_2^L(X_1, X_2, X_3; w, b),$$
$$\widehat{u}_3(X_1, X_2, X_3) = X_1\, \widehat{z}_3^L(X_1, X_2, X_3; w, b).$$

Result We also use FEM to simulate the 3D beam with a fine mesh to obtain the reference solution. We plot the displacement and stress along AB. Here, AB is the vertical centerline with the coordinates A(2, 0.5, 1) and B(2, 0.5, 0). Figure 18 shows the results obtained by DEMx compared with the reference solution. Again, DEMS and DEMT yield better results compared to DEMM. Figure 19 and Figure 20 show the contour plot of the vertical displacement and VonMises stress.

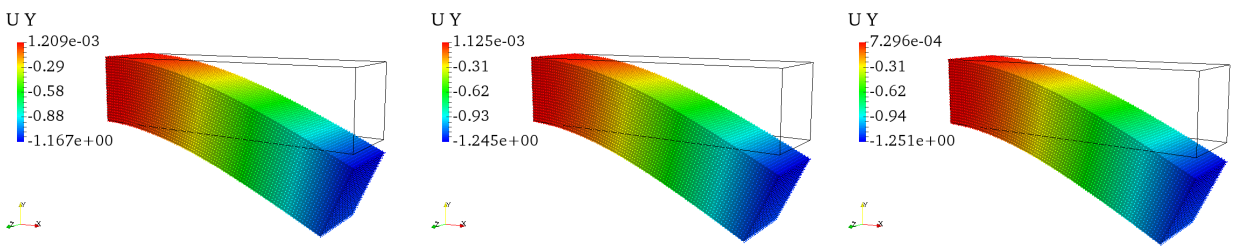
The error in the L^2 norm and H^1 seminorm can be found in Figures 21 and 22. Figure 21 shows that the approximate solution given by DEM converges with increasing training steps and DEMS yields the most accurate result. As we can see from Figure 22, the approximate solution converges with increasing size of the dataset.



(a) Displacement magnitude comparison

(b) Stress comparison

Figure 18: The displacement magnitude and stress comparison between DEM and the reference solution at AB in 3D bending beam test.

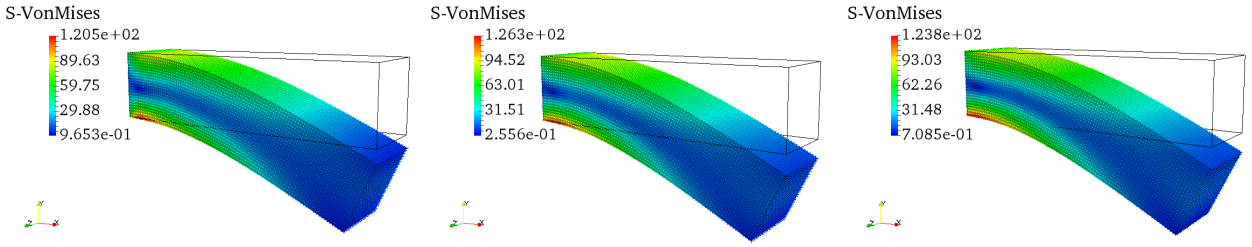


(a) u_2 - DEMM

(b) u_2 - DEMT

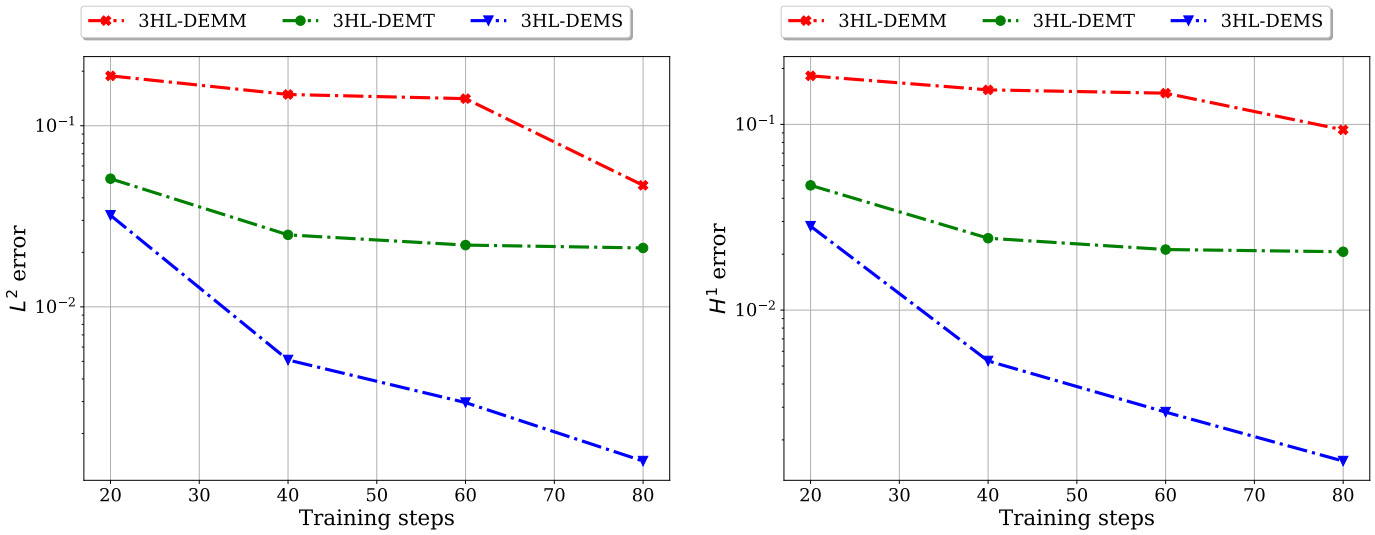
(c) u_2 - DEMS

Figure 19: The vertical displacements of all versions of DEM.



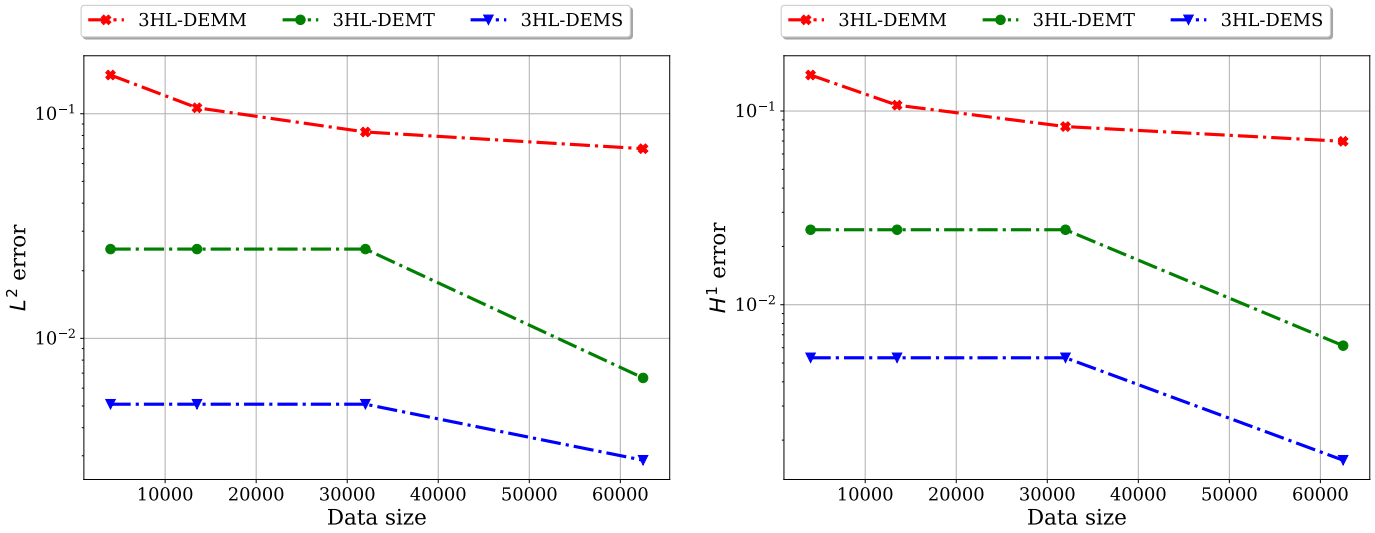
(a) VonMises stress - DEMM (b) VonMises stress - DEMT (c) VonMises stress - DEMS

Figure 20: The VonMises stresses of all versions of DEM.



(a) L^2 norm error (b) H^1 seminorm error

Figure 21: The error in terms of L^2 norm and H^1 seminorm of DEMx with respect to the training steps.



(a) L^2 norm error

(b) H^1 seminorm error

Figure 22: The error in terms of L^2 norm and H^1 seminorm of DEMx with respect to the size of dataset.

4.4. Twisting a Hyperelastic 3D Cuboid

In this example we study a cube made of hyperelastic materials under torsion. Inspired by [44] we adopt an example from Fenics [45]. A 3D cuboid made of hyperelastic material has a length $L = 1.25$ m, width $W = 1$ m, and depth $D = 1$ m. The cuboid is twisted by an angle of 60^o through a prescribed Dirichlet boundary condition $\boldsymbol{u}_{|\Gamma_1}$ at the right end of the solid and is fixed at the left end by $\boldsymbol{u}_{|\Gamma_0}$ (see Figure 23) given by

$$\begin{aligned} \boldsymbol{u}_{|\Gamma_0} &= [0,0,0]^T, \\ \boldsymbol{u}_{|\Gamma_1} &= \begin{bmatrix} 0 \\ 0.5\left[0.5 + (X_2 - 0.5)\cos(\pi/3) - (X_3 - 0.5)\sin(\pi/3) - X_2\right] \\ 0.5\left[0.5 + (X_2 - 0.5)\sin(\pi/3) + (X_3 - 0.5)\cos(\pi/3) - X_3\right] \end{bmatrix}. \end{aligned} \tag{43}$$

Four surrounding surfaces are subjected to surface-traction forces $\bar{\boldsymbol{t}} = [1,0,0]^T$. We also consider the body force $\boldsymbol{f}_b = [0,-0.5,0]^T$ of the solid.

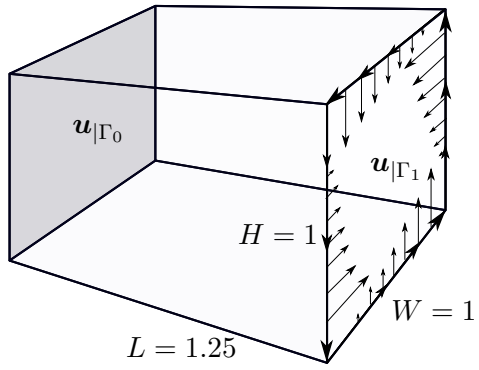


Figure 23: A Hyperelastic 3D Cuboid is twisted an angle of 60^o.

A Neo-Hookean constitutive model described in Equation (6) is adopted with the material properties as depicted in Table 2,

Description	Value
E - Young modulus [N/m^2]	10^6
ν - Poisson ratio	0.3
κ - bulk modulus [Pa]	$\frac{E}{3(1-2\nu)}$
μ - Lame parameter [Pa]	$\frac{E}{2(1+\nu)}$
λ - Lame parameter [Pa]	$\frac{E\nu}{(1+\nu)(1-2\nu)}$

Table 2: Material parameters for 3D hyperelastic Cuboid.

Setup We generate 64000 equidistant grid points ($N_1 = 40, N_2 = 40, N_3 = 40$) over the entire domain as the feeding data to train our network (Figure 24). A 5-layer network ($3 - 30 - 30 - 30 - 3$) is constructed. Therein, we use 3 neurons in the input layer for the nodal coordinates and 3 neurons in the output layer for the unconstrained solution. Moreover, in each hidden layer 30 neurons are used to enforce the learning. We use *tanh* activation function to evaluate neural values in the hidden layers. The learning rate $lr = 0.5$ is used in the L-BFGS optimizer. The neural network is strained for 50 steps. With the prescribed boundary conditions, the components of the trial function have the form

$$\widehat{\boldsymbol{u}}(\boldsymbol{X}) = \boldsymbol{u}_{|\Gamma_1} \frac{X_1}{1.25} + (X_1 - 1.25)\, X_1\, \widehat{\boldsymbol{z}}^L(\boldsymbol{X}; \boldsymbol{\theta}). \tag{44}$$

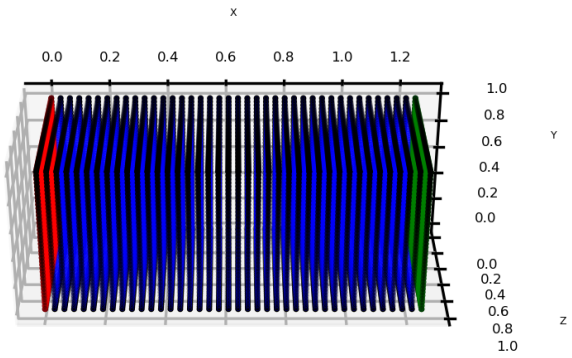
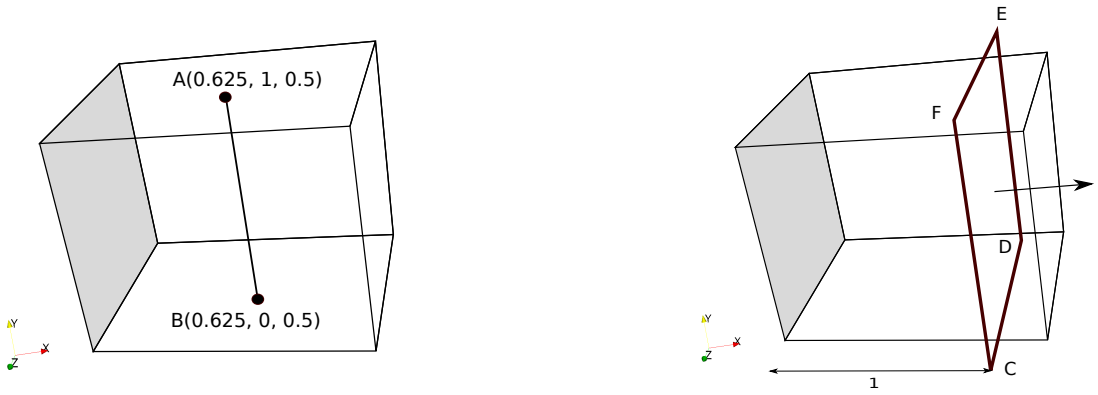


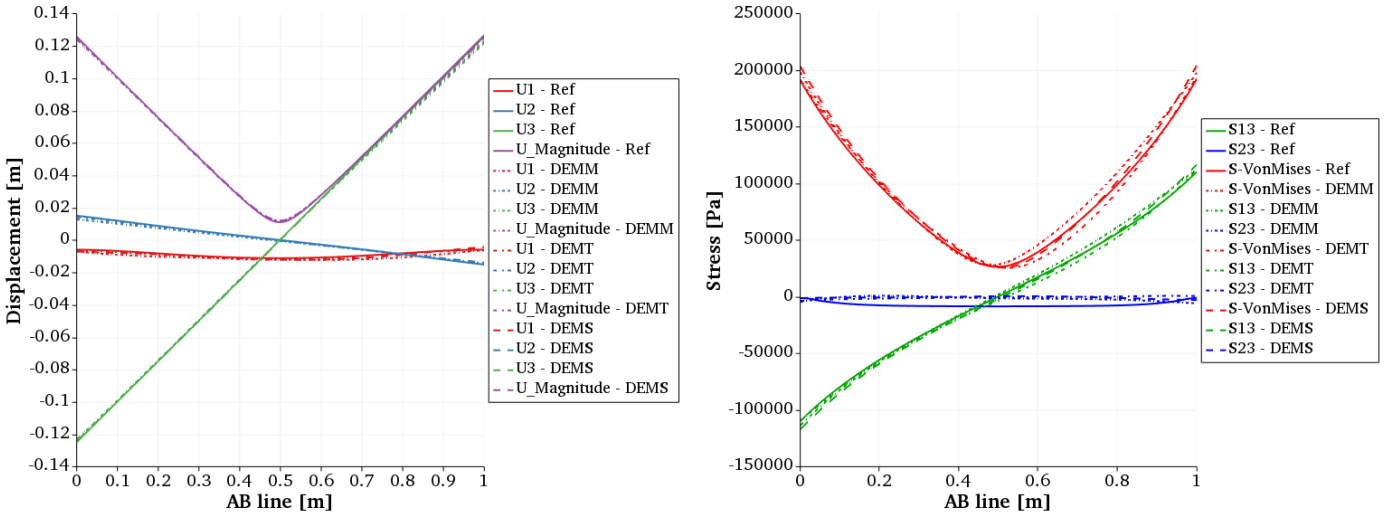
Figure 24: The training point distribution of a 3D hyperelastic Cuboid for the training step in DEM. The red points are used to enforce/train the first Dirichlet boundary condition. The green points are used to enforce/train the second Dirichlet boundary condition. The black points are used to train the surface force. The blue points are used for the interior domain integration and the body force.

Result A finite element solution on a fine mesh serves again as the reference solution. We first plot the displacements and stresses at the AB line depicted in Figure 25a. A good agreement between all three DEM versions and the reference solution is obtained, see Figure 26.



(a) AB line (b) CDEF plane

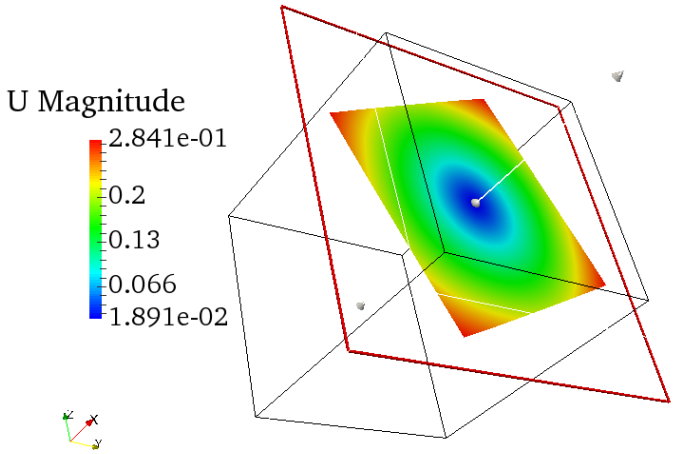
Figure 25: Positions of AB line and CDEF plane.



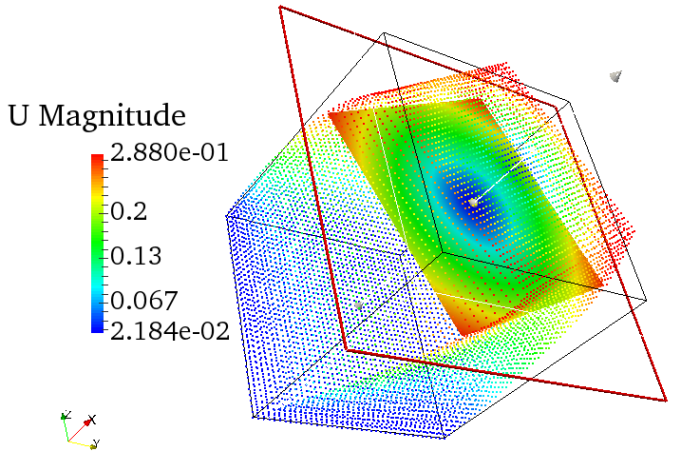
(a) Displacement comparison (b) Stress comparison

Figure 26: The displacement and stress results of DEMM, DEMT, DEMS measured at the AB line compared to the reference solution.

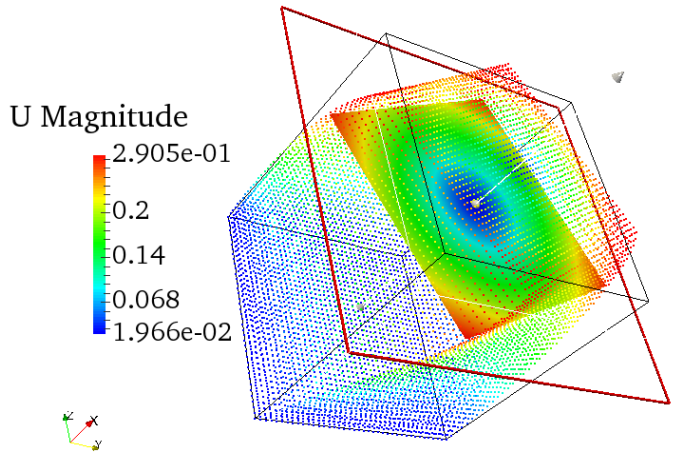
The magnitude of the displacement (magnitude of the 3D displacement vector) and the associated VonMises stress at the CDEF plane (25b) can be found in Figures 27 and 28.



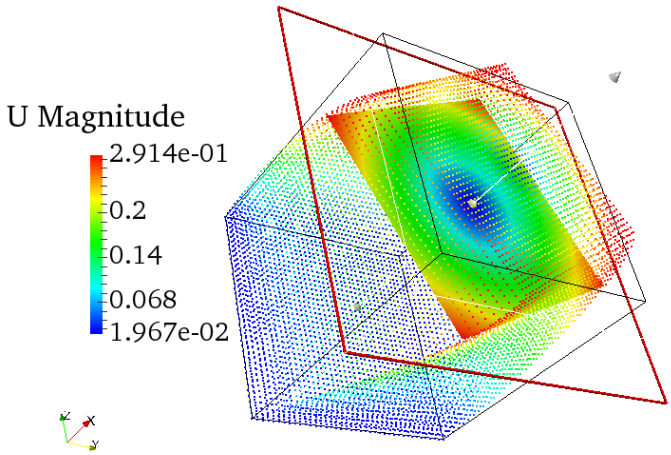
(a) FEM



(b) DEMM

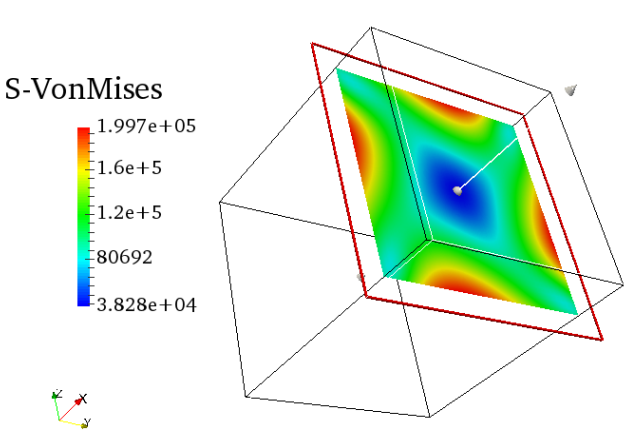


(c) DEMT

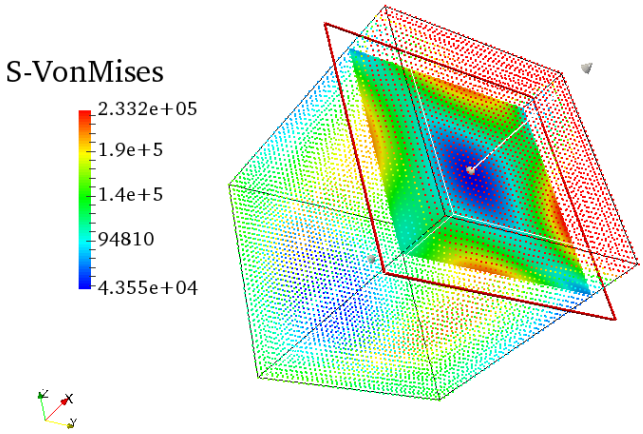


(d) DEMS

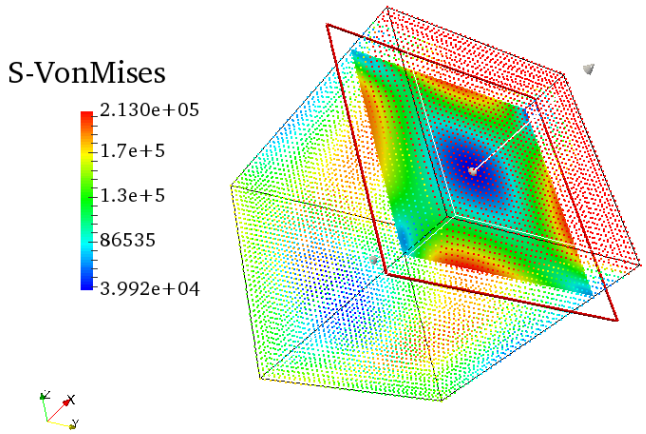
Figure 27: A comparison of displacement magnitude at the CDEF plane between reference solution and DEMx of a twisted Neo-Hookean 3D Cuboid.



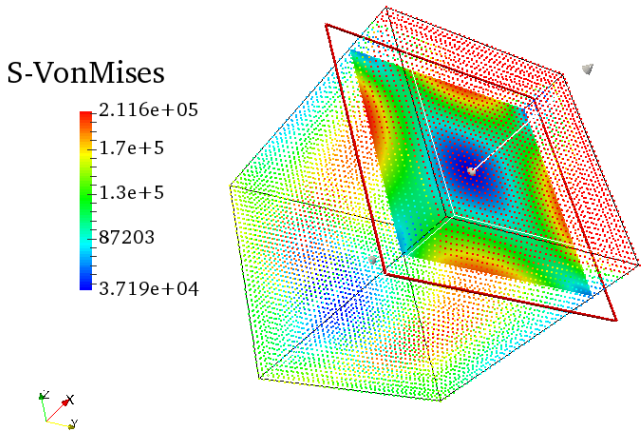
(a) FEM



(b) DEMM



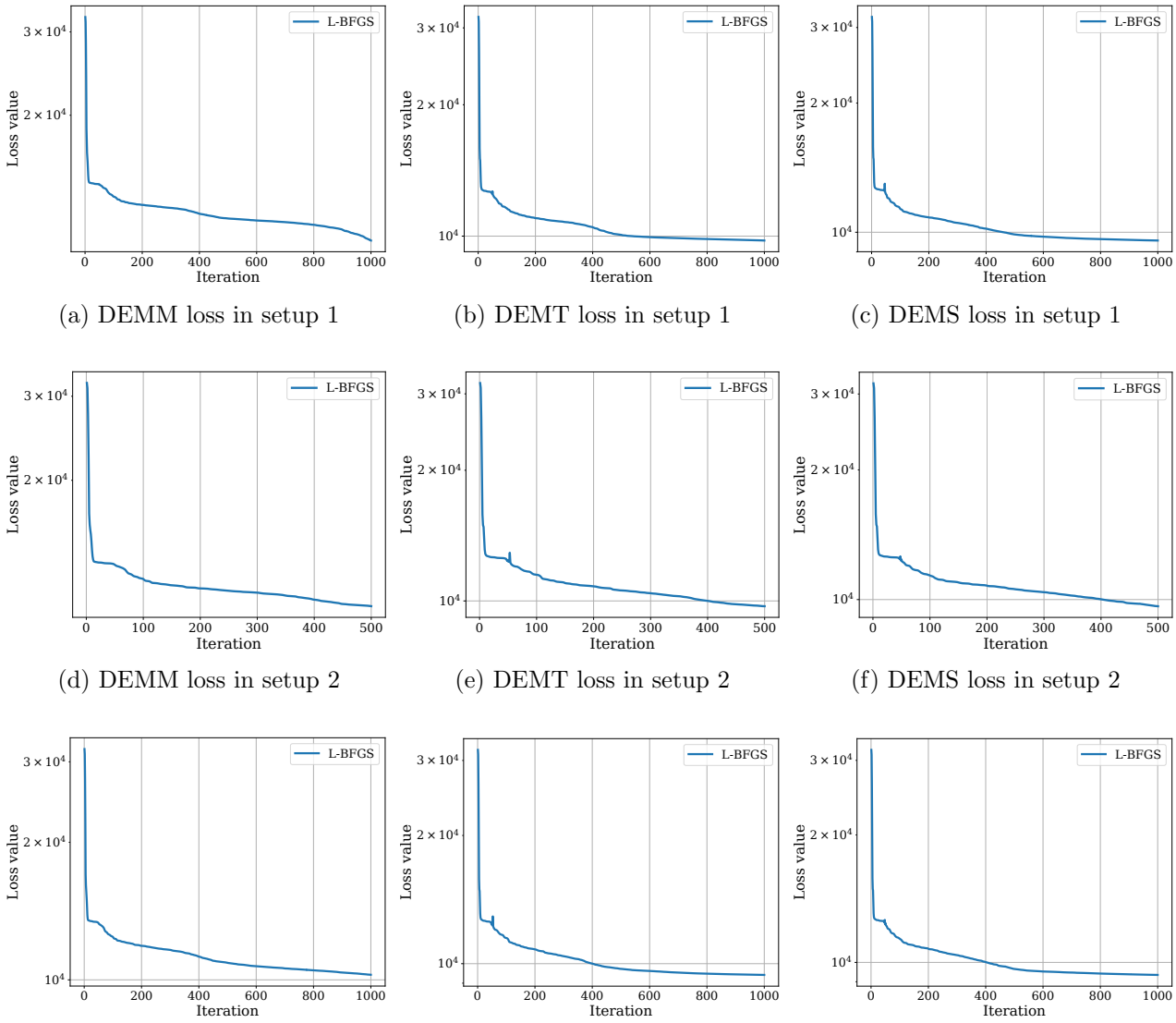
(c) DEMT



(d) DEMS

Figure 28: A comparison of VonMises stress at the CDEF plane between reference solution and DEMx of a twisted Neo-Hookean 3D Cuboid.

Let us consider the following setups: Setup 1: 8000 equidistant data points ($N_1 = 20, N_2 = 20, N_3 = 20$) where the network is trained for 50 steps. Setup 2: 64000 equidistant data points ($N_1 = 40, N_2 = 40, N_3 = 40$) and the neural network is trained for 25 steps. Setup 3: 64000 equidistant data points ($N_1 = 40, N_2 = 40, N_3 = 40$) and the neural network is trained for 50 steps. The L^2 norm and H^1 seminorm of the reference solution obtained by Fenics on a fine mesh are given as: $||u||^{FE}_{L^2} = 0.13275$ and $||u||^{FE}_{H^1} = 0.51407$. As shown in Table 3, DEMx obtain good results in terms of the error in the L^2 norm and H^1 seminorm. DEMT and DEMS yield more accurate results than DEMM in all three cases. Moreover, longer training steps lead to better results. The plots in Figure 29 illustrate the convergence of the loss functions of DEMx for the different setups.



(a) DEMM loss in setup 1 (b) DEMT loss in setup 1 (c) DEMS loss in setup 1

(d) DEMM loss in setup 2 (e) DEMT loss in setup 2 (f) DEMS loss in setup 2

(g) DEMM loss in setup 3 (h) DEMT loss in setup 3 (i) DEMS loss in setup 3

Figure 29: The convergence of loss functions of DEMM, DEMT, DEMS in different setups. Setup 1: We use 8000 equidistant data points ($N_1 = 20, N_2 = 20, N_3 = 20$) as the feeding data to train our network; the network is trained for 50 steps. Setup 2: We use 64000 equidistant data points ($N_1 = 40, N_2 = 40, N_3 = 40$) as the feeding data to train our network; the neural network is trained for 25 steps. Setup 3: We use 64000 equidistant data points ($N_1 = 40, N_2 = 40, N_3 = 40$) as the feeding data to train our network; the neural network is trained for 50 steps.

Data	DEMM		DEMT		DEMS	
	$\|\|u\|\|_{L^2}$	$\|\|u\|\|_{H^1}$	$\|\|u\|\|_{L^2}$	$\|\|u\|\|_{H^1}$	$\|\|u\|\|_{L^2}$	$\|\|u\|\|_{H^1}$
20x20x20 (50 steps)	0.12921	0.49929	0.13218	0.50991	0.13252	0.51143
40x40x40 (25 steps)	0.13210	0.51001	0.13216	0.51060	0.13304	0.51314
40x40x40 (50 steps)	0.13185	0.50872	0.13237	0.51131	0.13256	0.51253

Table 3: The L^2 norm and H^1 seminorm of DEMx in 3 setups. The corresponding norms of the reference solution are given by Fenics program with a fine mesh as: $||u||^{FE}_{L^2} = 0.13275$, $||u||^{FE}_{H^1} = 0.51407$.

4.5. Twisting a T-Structure

The aim of this example is to demonstrate the ability of DEM in solving a -geometrically-complex structure made of a hyperelastic material. This example has been solved by the finite element method in [46, 47]. The T-structure is fixed at the bottom. The vertical part is subjected to two opposite uniform pressure loads $p = 24$ N at two surfaces as shown in Figure 30. A Neo-Hookean model is employed represented by eq. (6) with bulk modulus $\kappa = 500/3$ Pa and shear modulus $\mu = 100$ Pa. The first Lame parameter is determined by the relation $\lambda = \kappa - 2\mu/3$.

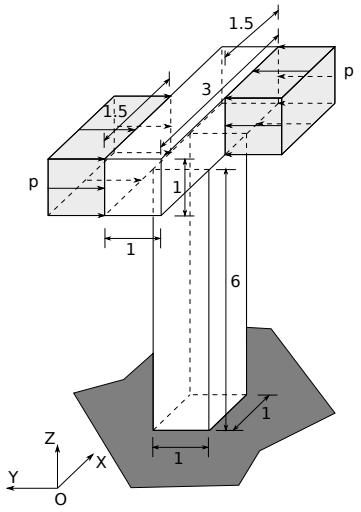


Figure 30: A T-structure which is made of homogeneous, hyperelastic material is twisted by 2 opposite loadings in 3D.

Setup We generate $10\times10\times60$ and $30\times10\times10$ points for the vertical and horizontal part, respectively. These points are considered as the feeding data to train our network (Figure 31). A 5-layer network ($3-30-30-30-3$) is constructed. Therein, we use 3 neurons in the input layer for the nodal coordinates and 3 neurons in the output layer for the unconstrained solution. Moreover, 30 neurons in each hidden layers are used to enforce the learning. *tanh* activation function is adopted to evaluate neural values in the hidden layers. The learning rate $lr = 0.5$ is used in the L-BFGS optimizer. The neural network is strained for 25 steps. With the prescribed boundary conditions, the components of the trial function have the form

$$\widehat{u}_1(X_1, X_2, X_3) = X_3\, \widehat{z}^L_1(X_1, X_2, X_3; w, b),$$
$$\widehat{u}_2(X_1, X_2, X_3) = X_3\, \widehat{z}^L_2(X_1, X_2, X_3; w, b),$$
$$\widehat{u}_3(X_1, X_2, X_3) = X_3\, \widehat{z}^L_3(X_1, X_2, X_3; w, b).$$

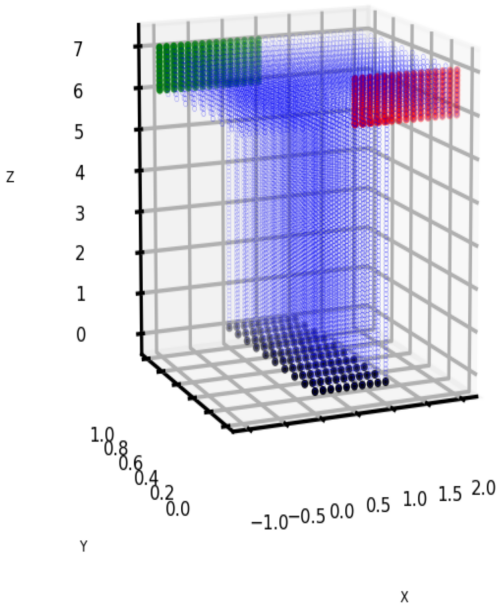
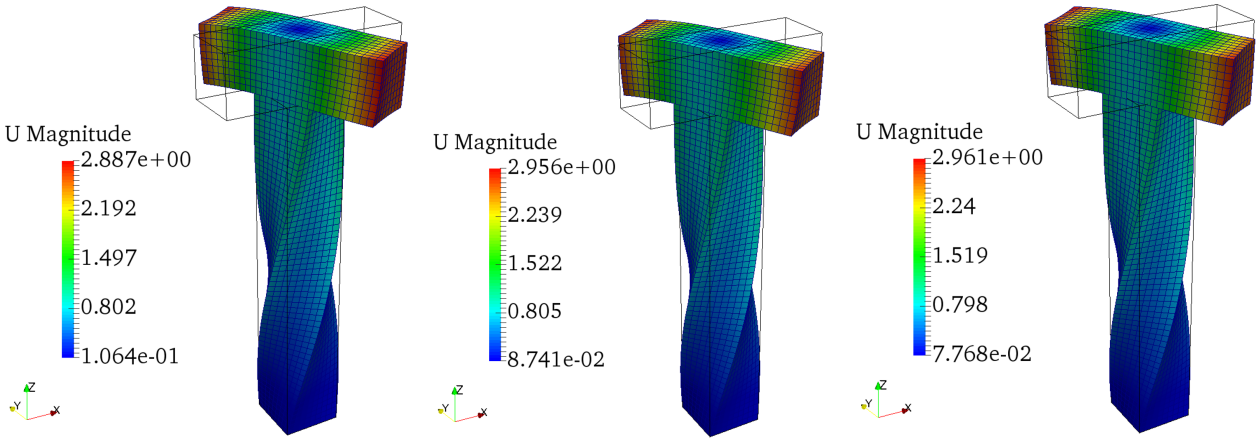


Figure 31: The training point distribution of a T-structure for the training step in DEM. The black points are used to enforce/train the first Dirichlet boundary condition. The green and red points are used to enforce/train the pressure applying. The blue points are used for the interior domain integration.

Result We use the same data in the training step for the predicting step. Figure 32 illustrates the displacement plots of the twisted T-structure given by DEM. The result is similar to the FEM solution [47]. Promisingly, with a pretty small dataset for a complex problem and lightweight settings, DEM is able to yield very good results in a very short time (138s).



(a) DEMM (b) DEMT (c) DEMS

Figure 32: The deformation of T-structure under two opposite pressure in DEMx.

5. Conclusion

In this work, we have presented a Deep Energy Method (DEM) for nonlinear large deformation hyperelasticity using deep neural networks. An energy functional is minimized in order to train the deep neural network. It uses the potential energy as a loss function and a 'standard' optimizer. The performance and efficiency

of the method are demonstrated through several benchmark problems including 2D and 3D bending beam problems and structures subjected to torsional loading. The method seems a promising alternative to classical methods such as FEM, especially for problems with high dimensionality – as they occur for instance in coupled problems, ill-posed problems in inverse analysis or stochastic PDEs, just to name a few. Once the network is trained for a wider class of boundary conditions and geometries, a solution of the underlying problem might be achieved in a much shorter time compared to classical methods. On the downside, DEM still requires improvements concerning the imposition of boundary conditions and integration techniques. Furthermore, more theoretical investigations on how to set up the deep neural network are required. This will be the topic of our future work.

6. Acknowledgment

The first and second authors owe the gratitude to the sponsorship from Sofja Kovalevskaja Programme of Alexander von Humboldt Foundation. The first author also would like to thank MSc. Somdatta Goswami and especially Dr. Cosmin Anitescu, Dr. Simon Hoell for the first code and the fruitful discussions during his research stay at Bauhaus Universität Weimar.

7. Appendix

7.1. Appendix A

Link to download code: `https://github.com/MinhNguyenIKM/dem_hyperelasticity`

7.2. Appendix B

The L^2 norm and H^1 seminorm of a vector $\boldsymbol{u}$ are defined as follows

$$||\boldsymbol{u}||_{L^2} = \sqrt{\int_B \boldsymbol{u}\cdot\boldsymbol{u}\, dV}, \tag{45}$$

$$||\boldsymbol{u}||_{H^1} = \sqrt{\int_B \nabla\boldsymbol{u} : \nabla\boldsymbol{u}\, dV}. \tag{46}$$

The derivation of first Piola-Kirchhoff stress based on the constitutive equation $\partial\Psi/\partial\boldsymbol{F}$

$$\textbf{Neo-Hookean:} \qquad \Psi(I_1, J) = \frac{1}{2}\lambda\left[\log(J)\right]^2 - \mu\log(J) + \frac{1}{2}\mu\left(I_1 - 3\right), \tag{47}$$

$$\begin{aligned} P_{iJ}\boldsymbol{e}_i\otimes\boldsymbol{e}_J &= \frac{\partial\Psi}{\partial F_{iJ}}\boldsymbol{e}_i\otimes\boldsymbol{e}_J \\ &= \left[\frac{1}{2}\lambda\frac{\partial[\log(J)]^2}{\partial J}\frac{\partial J}{\partial F_{iJ}} - \mu\frac{\partial\log(J)}{\partial J}\frac{\partial J}{\partial F_{iJ}} + \frac{1}{2}\mu\frac{\partial(I_1-3)}{\partial I_1}\frac{\partial I_1}{\partial F_{iJ}}\right]\boldsymbol{e}_i\otimes\boldsymbol{e}_J \\ &= \left[\lambda\log(J)\frac{1}{J}\frac{\partial\det(\boldsymbol{F})}{\partial F_{iJ}} - \mu\frac{1}{J}\frac{\partial\det(\boldsymbol{F})}{\partial F_{iJ}} + \frac{1}{2}\mu\frac{\partial\,\mathrm{trace}(\boldsymbol{F}^T\cdot\boldsymbol{F})}{\partial F_{iJ}}\right]\boldsymbol{e}_i\otimes\boldsymbol{e}_J, \\ &\text{with: } J = \det(\boldsymbol{F}) \qquad \text{and} \quad I_1 = \mathrm{trace}(\boldsymbol{F}^T\cdot\boldsymbol{F}) = \boldsymbol{F}:\boldsymbol{F}. \end{aligned} \tag{48}$$

We calculate $\partial_{\boldsymbol{F}} \det(\boldsymbol{F})$

$$\begin{aligned}\frac{\partial \det(\boldsymbol{F})}{\partial F_{iJ}} &= \frac{\partial \sum_L F_{iL} adj^T(\boldsymbol{F})_{iL}}{\partial F_{iJ}} = \sum_L \frac{\partial F_{iL}}{\partial F_{iJ}} adj^T(\boldsymbol{F})_{iL} + \sum_L F_{iL} \underbrace{\frac{\partial adj^T(\boldsymbol{F})_{iL}}{\partial F_{iJ}}}_{=0} \\ &= \sum_L \delta_{JL} adj^T(\boldsymbol{F})_{iL} = adj^T(\boldsymbol{F})_{iJ} = F^{-1}_{Ji} det(\boldsymbol{F}) = F^{-1}_{Ji} J \\ &\text{where} \quad adj(F)_{iJ} = F^{-1}_{iJ} det(\boldsymbol{F}).\end{aligned} \tag{49}$$

We calculate $\partial_{\boldsymbol{F}} \operatorname{trace}(\boldsymbol{F}^T \cdot \boldsymbol{F})$

$$\frac{\partial F_{kL} F_{kL}}{\partial F_{iJ}} = \frac{\partial F_{kL}}{\partial F_{iJ}} F_{kL} + F_{kL} \frac{\partial F_{kL}}{\partial F_{iJ}} = \delta_{ik}\delta_{JL} F_{kL} + F_{kL}\delta_{ik}\delta_{KL} = 2F_{iJ}. \tag{50}$$

We substitute the results 49 and 50 to 48

$$\begin{aligned}P_{iJ}\boldsymbol{e}_i \otimes \boldsymbol{e}_J &= \left[\lambda \log(J) F^{-1}_{Ji} - \mu F^{-1}_{Ji} + \mu F_{iJ}\right] \boldsymbol{e}_i \otimes \boldsymbol{e}_J \\ &= \left[\mu F_{iJ} + (\lambda \log(J) - \mu) F^{-T}_{iJ}\right] \boldsymbol{e}_i \otimes \boldsymbol{e}_J.\end{aligned} \tag{51}$$

Finally, we obtain

$$\boldsymbol{P} = \mu \boldsymbol{F} + [\lambda \log(J) - \mu] \boldsymbol{F}^{-T}. \tag{52}$$

The derivation for second Piola-Kirchhoff stress for the compressible Mooney-Rivlin model

$$\textbf{Mooney-Rivlin:} \qquad \Psi(I_1, I_2, J) = c(J-1)^2 - d\log(J) + c_1(I_1 - 3) + c_2(I_2 - 3), \tag{53}$$

where

$$I_1 = \operatorname{trace}(\boldsymbol{C}), \qquad I_2 = \frac{1}{2}[\operatorname{trace}(\boldsymbol{C})^2 - \operatorname{trace}(\boldsymbol{C}^2)], \qquad J = \sqrt{\det(\boldsymbol{C})}. \tag{54}$$

and

$$\begin{aligned}&\boldsymbol{C} = \boldsymbol{F}^T \cdot \boldsymbol{F} \\ \Rightarrow \quad &C_{JK} = F_{iJ}\boldsymbol{e}_J \otimes \boldsymbol{e}_i \cdot F_{kL}\boldsymbol{e}_k \otimes \boldsymbol{e}_L = F_{iJ}F_{iL}\boldsymbol{e}_J \otimes \boldsymbol{e}_K\end{aligned} \tag{55}$$

$$\begin{aligned}S_{IJ}\boldsymbol{e}_I \otimes \boldsymbol{e}_J &= 2\frac{\partial \Psi}{\partial C_{IJ}} \boldsymbol{e}_I \otimes \boldsymbol{e}_J \\ &= 2\left[2c(J-1)\frac{\partial J}{\partial C_{IJ}} - d\frac{1}{J}\frac{\partial J}{\partial C_{IJ}} + c_1 \frac{\partial I_1}{\partial C_{IJ}} + c_2 \frac{\partial I_2}{\partial C_{IJ}}\right] \boldsymbol{e}_I \otimes \boldsymbol{e}_J.\end{aligned} \tag{56}$$

We calculate $\partial_{\boldsymbol{C}} J$

$$\begin{aligned}\frac{\partial \sqrt{\det(\boldsymbol{C})}}{C_{IJ}} &= \frac{1}{2\sqrt{\det(\boldsymbol{C})}} \frac{\partial \det(\boldsymbol{C})}{C_{IJ}} \\ &= \frac{1}{2\sqrt{\det(\boldsymbol{C})}} C^{-1}_{JI} \det(\boldsymbol{C}) = \frac{1}{2} J C^{-T}_{IJ}.\end{aligned} \tag{57}$$

We calculate $\partial_{\boldsymbol{C}} I_1$

$$\begin{aligned}\frac{\partial \operatorname{trace}(\boldsymbol{C})}{\partial C_{IJ}} &= \frac{\partial (C_{KL}\delta_{KL})}{\partial C_{IJ}} \\ &= \frac{\partial C_{KL}}{\partial C_{IJ}}\delta_{KL} = \delta_{IK}\delta_{JL}\delta_{KL} = \delta_{IJ}. \end{aligned} \tag{58}$$

We calculate $\partial_{\boldsymbol{C}} I_2$

$$\begin{aligned}\frac{\partial 1/2[\operatorname{trace}(\boldsymbol{C})^2 - \operatorname{trace}(\boldsymbol{C}^2)]}{\partial C_{IJ}} &= \frac{1}{2}\left[\frac{\partial \operatorname{trace}(\boldsymbol{C})^2}{\partial C_{IJ}} - \frac{\partial \operatorname{trace}(\boldsymbol{C}\cdot\boldsymbol{C})}{\partial C_{IJ}}\right] \\ &= \frac{1}{2}\left[2\operatorname{trace}(\boldsymbol{C})\delta_{IJ} - \frac{\partial (C_{KM}C_{ML}\delta_{KL})}{\partial C_{IJ}}\right] \\ &= \frac{1}{2}\left[2\operatorname{trace}(\boldsymbol{C})\delta_{IJ} - \delta_{IK}\delta_{JM}C_{ML}\delta_{KL} - C_{KM}\delta_{IM}\delta_{JL}\delta_{KL}\right] \\ &= \frac{1}{2}\left[2C_{MN}\delta_{MN}\delta_{IJ} - 2C_{JI}\right] = C_{MM}\delta_{IJ} - C_{JI}. \end{aligned} \tag{59}$$

We substitute the results (59), (58) and (57) into (56) to obtain

$$\begin{aligned}S_{IJ}\boldsymbol{e}_I \otimes \boldsymbol{e}_J &= 2\left[c(J-1)JC_{IJ}^{-T} - \frac{d}{2}C_{IJ}^{-T} + c_1\delta_{IJ} + c_2(C_{MM}\delta_{IJ} - C_{JI})\right]\boldsymbol{e}_I \otimes \boldsymbol{e}_J \\ &= \left[(2c_1 + 2c_2C_{MM})\delta_{IJ} - 2c_2C_{JI} + [2c(J-1)J - d]C_{IJ}^{-T}\right]\boldsymbol{e}_I \otimes \boldsymbol{e}_J. \end{aligned} \tag{60}$$

Finally, we obtain

$$\boldsymbol{S} = (2c_1 + 2c_2 I_1)\mathbf{1} - 2c_2\boldsymbol{C} + [2c(J-1)J - d]\boldsymbol{C}^{-1}. \tag{61}$$

References

[1] T. Belytschko, S. Loehnert, and J.-H. Song, "Multiscale aggregating discontinuities: A method for circumventing loss of material stability," *International Journal for Numerical Methods in Engineering*, vol. 73, no. 6, pp. 869–894, 2008.

[2] T. Belytschko and J.-H. Song, "Coarse-graining of multiscale crack propagation," *International Journal for Numerical Methods in Engineering*, vol. 81, no. 5, pp. 537–563, 2010.

[3] J.-H. Song and Y.-C. Yoon, "Multiscale failure analysis with coarse-grained micro cracks and damage," *Theoretical and Applied Fracture Mechanics*, vol. 72, pp. 100 – 109, 2014. Multiscale Modeling of Material Failure.

[4] J.-H. Song and T. Belytschko, "Multiscale aggregating discontinuities method for micro–macro failure of composites," *Composites Part B: Engineering*, vol. 40, no. 6, pp. 417 – 426, 2009. Blast/impact on engineered (nano)composite materials.

[5] A. Tabarraei, J.-H. Song, and H. Waisman, "A two-scale strong discontinuity approach for evolution of shear bands under dynamic impact loads," *International Journal for Multiscale Computational Engineering*, vol. 11, no. 6, pp. 543–563, 2013.

[6] Z. Liu, M. Bessa, and W. K. Liu, "Self-consistent clustering analysis: An efficient multi-scale scheme for inelastic heterogeneous materials," *Computer Methods in Applied Mechanics and Engineering*, vol. 306, pp. 319 – 341, 2016.

[7] J. Yvonnet and Q.-C. He, "The reduced model multiscale method (r3m) for the non-linear homogenization of hyperelastic media at finite strains," *Journal of Computational Physics*, vol. 223, no. 1, pp. 341 – 368, 2007.

[8] F. Fritzen and O. Kunc, "Two-stage data-driven homogenization for nonlinear solids using a reduced order model," *European Journal of Mechanics - A/Solids*, vol. 69, pp. 201 – 220, 2018.

[9] K. Hoang, Y. Fu, and J. Song, "An hp-proper orthogonal decomposition–moving least squares approach for molecular dynamics simulation," *Computer Methods in Applied Mechanics and Engineering*, vol. 298, pp. 548 – 575, 2016.

[10] K. C. Hoang, T.-Y. Kim, and J.-H. Song, "Fast and accurate two-field reduced basis approximation for parametrized thermoelasticity problems," *Finite Elements in Analysis and Design*, vol. 141, pp. 96 – 118, 2018.

[11] T. Kirchdoerfer and M. Ortiz, "Data-driven computational mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 304, pp. 81 – 101, 2016.

[12] R. Ibañez, D. Borzacchiello, J. V. Aguado, E. Abisset-Chavanne, E. Cueto, P. Ladeveze, and F. Chinesta, "Data-driven non-linear elasticity: constitutive manifold construction and problem discretization," *Computational Mechanics*, vol. 60, pp. 813–826, Nov 2017.

[13] L. T. K. Nguyen and M.-A. Keip, "A data-driven approach to nonlinear elasticity," *Computers & Structures*, vol. 194, pp. 97 – 115, 2018.

[14] M. Bessa, R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen, and W. Liu, "A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality," *Computer Methods in Applied Mechanics and Engineering*, vol. 320, pp. 633 – 667, 2017.

[15] B. A. Le, J. Yvonnet, and Q.-C. He, "Computational homogenization of nonlinear elastic materials using neural networks," *International Journal for Numerical Methods in Engineering*, vol. 104, no. 12, pp. 1061–1084, 2015.

[16] V. M. Nguyen-Thanh, L. T. K. Nguyen, T. Rabczuk, and X. Zhuang, "A surrogate model for computational homogenization of elastostatics at finite strain using the HDMR-based neural network approximator," *arXiv preprint arXiv:1906.02005*, 2019.

[17] K.-J. Bathe, *Finite Element Procedures*. Prentice Hall, 2006.

[18] R. W. Clough, *The Finite Element Method in Plane Stress Analysis*. American Society of Civil Engineers, 1960.

[19] R. W. Clough, "Original formulation of the finite element method," 1989.

[20] T. Belytschko, Y. Y. Lu, and L. Gu, "Element-free galerkin methods," *International Journal for Numerical Methods in Engineering*, vol. 37, no. 2, pp. 229–256, 1994.

[21] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, no. 1, pp. 543–574, 1992.

[22] V. P. Nguyen, T. Rabczuk, S. Bordas, and M. Duflot, "Meshless methods: A review and computer implementation aspects," *Mathematics and Computers in Simulation*, vol. 79, no. 3, pp. 763 – 813, 2008.

[23] T. Hughes, J. Cottrell, and Y. Bazilevs, "Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39, pp. 4135 – 4195, 2005.

[24] J. Cottrell, T. Hughes, and Y. Bazilevs, *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley, 2009.

[25] V. P. Nguyen, C. Anitescu, S. P. Bordas, and T. Rabczuk, "Isogeometric analysis: An overview and computer implementation aspects," *Mathematics and Computers in Simulation*, vol. 117, pp. 89 – 116, 2015.

[26] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, Sep. 1998.

[27] M. Raissi and G. E. Karniadakis, "Hidden physics models : Machine learning of nonlinear partial differential equations," *Journal of Computational Physics*, vol. 357, pp. 125–141, 2018.

[28] J. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339 – 1364, 2018.

[29] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *Neurocomputing*, vol. 317, pp. 28–41, 2018.

[30] H. Guo, X. Zhuang, and T. Rabczuk, "A deep collocation method for the bending analysis of kirchhoff plate," *Computers, Materials and Continua*, vol. 59, pp. 433–456, 2 2019.

[31] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk, "Artificial neural network methods for the solution of second order boundary value problems," *Computers, Materials and Continua*, vol. 59, pp. 345–359, 1 2019.

[32] E. Weinan and Bing Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, pp. 1–12, Mar 2018.

[33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[35] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, pp. 503–528, Aug 1989.

[36] R. W. Ogden, *Non-linear Elastic Deformations*. Dover Civil and Mechanical Engineering, Dover Publications, 1997.

[37] P. Wriggers, *Nonlinear Finite Element Methods*. Springer Berlin Heidelberg, 2008.

[38] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[39] M. Nielsen, "Neural Networks and Deep Learning," 2018.

[40] P. Jain and P. Kar, "Non-convex optimization for machine learning," *arXiv preprint arXiv:1712.07897*, 2017.

[41] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[42] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[43] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10566*, 2017.

[44] A. Logg, K.-A. Mardal, G. N. Wells, *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[45] C. Mavroyiakoumou, "Solving non-linear elasticity BVPs in FEniCS, bifurcation and buckling of beams," *A special topic report submitted for the degree of M.Sc. Mathematical Modelling and Scientific Computing, University of Oxford*, 2017.

[46] J. Bonet, A. J. Gil, and R. D. Wood, *Nonlinear Solid Mechanics for Finite Element Analysis: Statics*. Cambridge University Press, 2016.

[47] M. Guo and J. S. Hesthaven, "Reduced order modeling for nonlinear structural analysis using gaussian process regression," *Computer Methods in Applied Mechanics and Engineering*, vol. 341, pp. 807 – 826, 2018.