A deep learning framework for solution and discovery in solid mechanics: linear elasticity

Ehsan Haghighat[a], Maziar Raissi[b], Adrian Moure[c], Hector Gomez[c], Ruben Juanes[a]

[a]*Massachusetts Institute of Technology, Cambridge, MA*
[b]*University of Colorado Boulder, Boulder, CO*
[c]*Purdue University, West Lafayette, IN*

Abstract

We present the application of a class of deep learning, known as Physics Informed Neural Networks (PINN), to learning and discovery in solid mechanics. We explain how to incorporate the momentum balance and constitutive relations into PINN, and explore in detail the application to linear elasticity, although the framework is rather general and can be extended to other solid-mechanics problems. While common PINN algorithms are based on training one deep neural network (DNN), we propose a multi-network model that results in more accurate representation of the field variables. To validate the model, we test the framework on synthetic data generated from analytical and numerical reference solutions. We study convergence of the PINN model, and show that Isogeometric Analysis (IGA) results in superior accuracy and convergence characteristics compared with classic low-order Finite Element Method (FEM). We also show the applicability of the framework for transfer learning, and find vastly accelerated convergence during network re-training. Finally, we find that honoring the physics leads to improved robustness: when trained only on a few parameters, we find that the PINN model can accurately predict the solution for a wide range of parameters new to the network—thus pointing to an important application of this framework to sensitivity analysis and surrogate modeling.

Keywords: Artificial neural network, Deep learning, Physics-informed, Inversion, Sensitivity analysis, Transfer learning, Linear elasticity, SciANN

1. Introduction

Over the past few years, there has been a revolution in the successful application of Artificial Neural Networks (ANN), also commonly referred to Deep Neural Networks (DNN) and Deep Learning (DL), in various fields including image classification, handwriting recognition, speech recognition and translation, and computer vision. These ANN approaches have led to a sea change in the performance of search engines, autonomous driving, e-commerce, and photography (see [1, 2, 3] for a review). In engineering and science, ANNs have been applied to an increasing number of areas, including geosciences [4, 5, 6, 7, 8], material science [9, 10, 11, 12], fluid mechanics [13, 14], genetics [15], and infrastructure health monitoring [16, 17], to name a few examples. In the solid and geomechanics community, deep learning has been used primarily for material modeling, in an attempt to replace classical constitutive models with ANNs [18, 19, 20]. In these applications, training of the network, i.e., evaluation of the network parameters, is carried out by

minimizing the norm of the distance between the network output (prediction) and the true output (training data). In this paper, we will refer to ANNs trained in this way as "data-driven."

A different class of ANNs, known as Physics-Informed Neural Networks (PINN), was introduced recently [21, 22, 23, 24, 25]. This new concept of ANNs was developed to endow the network model with known equations that govern the physics of a system. The training of PINNs is performed with a cost function that, in addition to data, includes the governing equations, initial and boundary conditions. This architecture can be used for solution and discovery (finding parameters) of systems of ordinary differential equations (ODEs) and partial differential equations (PDEs). While solving ODEs and PDEs with ANNs is not a new topic, e.g., [26, 27, 28], the success of these new studies can be broadly attributed to: (1) the choice of network architecture, i.e., the set of inputs and outputs of the ANN, so that one can impose governing equations on the network; (2) algorithmic advances, including graph-based automatic differentiation for accurate differentiation of ANN functionals and for error back-propagation; and (3) availability of advanced machine-learning software with CPU and GPU parallel processing capabilities including Theano [29] and TensorFlow [30].

This framework has been used for solution and discovery of Schrodinger, Allen–Cahn, and Navier–Stokes equations [22, 21]. It has also been used for solution of high-dimensional stochastic PDEs [23]. As pointed out in [23], this approach can be considered as a class of Reinforcement Learning [31], where the learning is on maximizing an incentive or minimizing a loss rather than direct training on data. If the network prediction does not satisfy a governing equation, it will result in an increase in the cost and therefore the learning traverses a path that minimizes that cost.

Here, we focus on the novel application of PINNs to solution and discovery of solid mechanics. We focus on linear elasticity, but the proposed framework may be applied to other linear and nonlinear problems of solid mechanics. Since parameters of the governing PDEs can also be defined as trainable parameters, the framework inherently allows us to perform parameter identification (model inversion). We validate the framework on synthetic data generated from low-order and high-order Finite Element Methods (FEM) and from Isogeometric Analysis (IGA) [32, 33]. These datasets satisfy the governing equations with different order of accuracy, where the error can be considered as noise in data. We find that the training converges faster on more accurate datasets, pointing to importance of higher-order numerical methods for pre-training ANNs. We also find that if the data is pre-processed properly, the training converges to the correct solution and correct parameters even on data generated with a coarse mesh and low-order FEM—an important result that illustrates the robustness of the proposed approach. Finally, we find that, due to the imposition of the physics constraints, the training converges on a very sparse data set, which is a crucial property in practice given that the installation of a dense network of sensors can be very costly.

Parameter estimation (identification) of complex models is a challenging task that requires a large number of forward simulations, depending on model complexity and the number of parameters. As a result, most inversion techniques have been applied to simplified models. The use of PINNs, however, allows us to perform identification simultaneously with fitting the ANN model on data [22]. This property highlights the potential of this approach compared with classical methods. We explore the application of PINN models for identification of multiple datasets generated with different parameters. Similar to transfer learning, where a pre-trained model is used as the initial state of the network [34], we perform re-training on new datasets starting from a previously trained network on a different dataset (with different parameters). We find that the re-training and identification of other datasets take far less time. Since the successfully trained PINN model should also

satisfy the physics constraints, it is in effect a surrogate model that can be used for extrapolation on unexplored data. To test this property, we train a network on four datasets with different parameters and then test it on a wide range of new parameter sets, and find that the results remain relatively accurate. This property points to the applicability of PINN models for sensitivity analysis, where classical approaches typically require an exceedingly large number of forward simulations.

2. Physics-Informed Neural Networks: Linear Elasticity

In this section, we review the equations of linear elastostatics with emphasis on PINN implementation.

2.1. Linear elasticity

The equations expressing momentum balance, the constitutive model and the kinematic relations are, respectively,

$$\begin{aligned} \sigma_{ij,j} - f_i &= 0, \\ \sigma_{ij} &= \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}, \\ \varepsilon_{ij} &= \frac{1}{2}\left(u_{i,j} + u_{j,i}\right). \end{aligned} \tag{1}$$

Here, σ_{ij} denotes the Cauchy stress tensor. For the two-dimensional problems considered here $i, j = 1, 2$ (or $i, j = x, y$). We use the summation convention, and an subscript comma denotes partial derivative. The function f_i denotes a body force, u_i represents the displacements, ε_{ij} is the infinitesimal stress tensor and δ_{ij} is the Kronecker delta. The Lamé parameters λ and μ are the quantities to be inferred using PINN.

2.2. Introduction to Physics-Informed Neural Networks

In this section, we provide an overview of the Physics-Informed Neural Networks (PINN) architecture, with emphasis on their application to model inversion. Let $\mathcal{N}(\mathbf{x}; \mathbf{W}, \mathbf{b}) : \mathbb{R}^{d_\mathbf{x}} \to \mathbb{R}^{d_\mathbf{y}}$ be an L-layer neural network with input vector $\mathbf{x}$, output vector $\mathbf{y}$, and network parameters $\mathbf{W}, \mathbf{b}$. This network is a feed-forward network, meaning that each layer creates data for the next layer through the following nested transformations:

$$\mathbf{z}^l = \sigma^l\left(\mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}^l\right), \quad l = 1, \ldots, L, \tag{2}$$

where $\mathbf{z}^0 \equiv \mathbf{x}$ and $\mathbf{z}^L \equiv \mathbf{y}$ are inputs and outputs of the model, $\mathbf{W}^l, \mathbf{b}^l$ are parameters of each layer l, known as weights and biases, respectively. The functions σ^l are called activation functions and make the network nonlinear with respect to the inputs. For instance, an ANN functional of some field variable, such as displacement $u(\mathbf{x})$, with three hidden layers and with $\sigma^l = \tanh$ as the activation function for all layers except the last can be written as

$$\begin{aligned} \mathbf{z}^1(\mathbf{x}) &= \tanh(\mathbf{W}^0 \mathbf{x} + \mathbf{b}^0), \\ \mathbf{z}^2(\mathbf{x}) &= \tanh(\mathbf{W}^1 \mathbf{z}^1 + \mathbf{b}^1), \\ \mathbf{z}^3(\mathbf{x}) &= \tanh(\mathbf{W}^2 \mathbf{z}^2 + \mathbf{b}^2), \\ u(\mathbf{x}) &= \mathbf{W}^3 \mathbf{z}^3 + \mathbf{b}^3. \end{aligned} \tag{3}$$

This model can be considered as an approximate solution for the field variable u of a partial differential equation.

In the PINN architecture, the network inputs (also known as features) are space and time variables, i.e., (x, y, z, t) in Cartesian coordinates, which makes it meaningful to perform the differentiation of the network's output with respect to any of the input variables. Classical implementations based on finite difference approximations are not accurate when applied to deep networks (see [35] for a review). Thanks to modern graph-based implementation of the feed-forward network (e.g., Theano [29], Tensorflow [30], MXNet [36]), this can be carried out using Automatic Differentiation at machine precision, therefore allowing for many hidden layers to represent nonlinear response. Hence, evaluation of a partial differential operator $\mathcal{P}$ acting on u is achieved naturally with graph-based differentiation and can then be incorporated in the cost function along with initial and boundary conditions as:

$$\mathcal{L} = |u - u^*| + |\mathcal{P}u - 0^*| + |u - u^*|_{\partial\Omega} + |u_0 - u_0^*|, \tag{4}$$

where $\partial\Omega$ is the domain boundary, $u_0 - u_0^*$ is the initial condition at $t = t_0$, and 0^* indicates the expected (true) value for the differential relation $\mathcal{P}u$ at any given training point. The norm $|\cdot|$ of a generic quantity g defined in Ω denotes $\frac{1}{N}\sum_{i=1}^{N} g(\mathbf{x}_i)^2$ where the $\mathbf{x}_i$'s are the spatial points where the data is known. The dataset is then fed to the neural network and an optimization is performed to evaluate all the parameters of the model, including the parameters of the PDE.

2.3. Training PINN

Different algorithms that can be used to train a neural network. Among the choices available in Keras [37] we use the Adam optimization scheme [38], which we have found to outperform other choices such as Adagrad [39], for this task. Several algorithmic parameters affect the rate of convergence of the network training. Here we adopt the terminology in Keras [37], but the terminology in other modern machine learning packages is similar. The algorithmic parameters include *batch-size*, *epochs*, *shuffle*, and *patience*. Batch-size controls the number of samples from a dataset used to evaluate one gradient update. A batch-size of 1 would be associated with a full stochastic gradient descent optimization. One epoch is one round of training on a dataset. If a dataset is shuffled, then a new round of training (epoch) would result in an updated parameter set because the batched-gradients are evaluated on different batches. It is common to re-shuffle a dataset many times and perform the back-propagation updates. The optimizer may, however, stop earlier if it finds that new rounds of epochs are not improving the cost function. That is where the last keyword, patience, comes in. This is mainly because we are dealing with non-convex optimization and we need to test the training from different starting points and in different directions to build confidence on the parameters evaluated from minimization of the cost-function on a dataset. Patience is the parameter that controls when the optimizer should stop the training.

There are three ways to train the network: (1) generate a sufficiently large number of datasets and perform a one-epoch training on each dataset, (2) work on one dataset over many epochs by reshuffling the data, and (3) a combination of these. When dealing with synthetic data, all approaches are feasible to pursue. However, strategy (1) above is usually impossible to apply in practice, specially in space, where sensors are installed at fixed and limited locations. In the original work on PINN [22], approach (1) was used to train the model, where datasets are generated on random space discretizations at each epoch. Here, we follow approach (2) to use training data

that we could realistically have in practice. For all examples, unless otherwise noted, we use a batch-size of 64, a limit of 10,000 epochs with shuffling, and a patience of 500 to perform the training.

3. Illustrative example and discussions

In this section, we use the PINN architecture on an illustrative linear elasticity problem.

3.1. Problem setup

To illustrate the application of the proposed approach, we consider an elastic plane-strain problem on the unit square (Fig. 1), subject to the boundary conditions depicted in the figure. The body forces are:

$$\begin{aligned} f_x &= \lambda \left[-4\pi^2 \cos(2\pi x)\sin(\pi y) + \pi \cos(\pi x) Q y^3\right] \\ &\quad + \mu \left[-9\pi^2 \cos(2\pi x)\sin(\pi y) + \pi \cos(\pi x) Q y^3\right] \\ f_y &= \lambda \left[3\sin(\pi x) Q y^2 - 2\pi^2 \sin(2\pi x)\cos(\pi y)\right] \\ &\quad + \mu \left[6\sin(\pi x) Q y^2 - 2\pi^2 \sin(2\pi x)\cos(\pi y) - \pi^2 \sin(\pi x) Q y^4/4\right]. \end{aligned} \tag{5}$$

The exact solution of this problem is

$$u_x(x,y) = \cos(2\pi x)\sin(\pi y), \tag{6}$$

$$u_y(x,y) = \sin(\pi x) Q y^4/4. \tag{7}$$

which is plotted in Fig. 2, for parameter values of $\lambda = 1$, $\mu = 0.5$, and $Q = 4$.

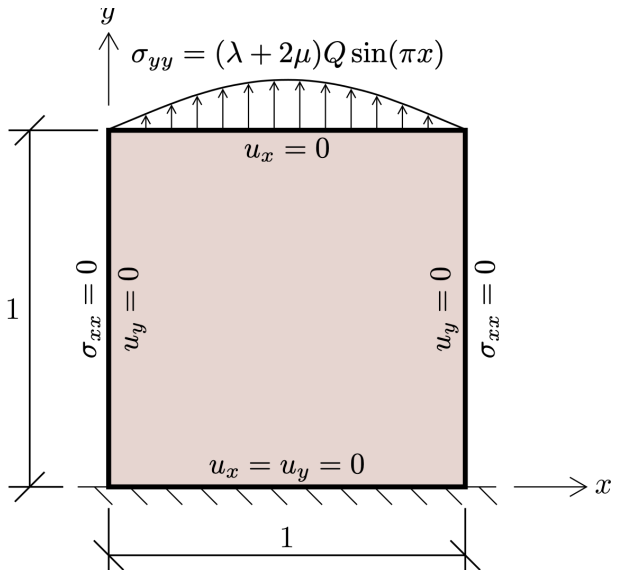

Figure 1: Problem setup and boundary conditions.

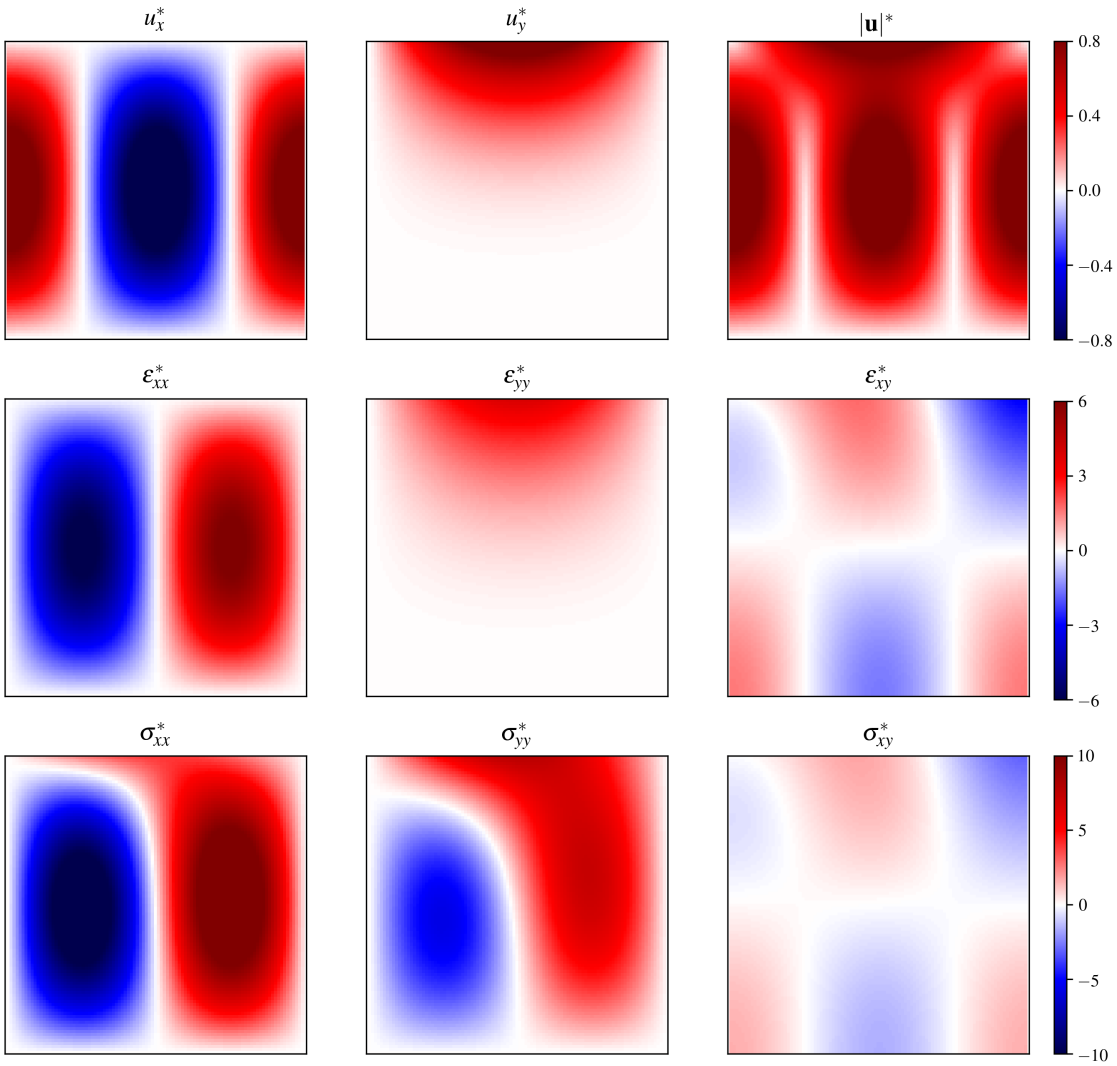

Figure 2: Exact solution in Eqs. (6)–(7) for parameter values of $\lambda = 1$, $\mu = 0.5$, and $Q = 4$.

3.2. Neural Network Setup

Due to the symmetry of the stress and strain tensors, the quantities of interest for a two-dimensional problem are u_x, u_y, ε_{xx}, ε_{yy}, ε_{xy}, σ_{xx}, σ_{yy}, σ_{xy}. There are a few potential architectures that we can use to design our network. The input features (variables) are the spatial coordinates (x, y), for all the network choices. For the outputs, a potential design is to have a densely connected network with two outputs as (u_x, u_y). Another option is to have two densely connected independent networks with only one output each, associated with u_x and u_y, respectively (Fig. 3). Then, the remaining quantities of interest, i.e., $\sigma_{ij}, \varepsilon_{ij}$, can be obtained through differentiation. Alternatively, we may have $(u_x, u_y, \sigma_{xx}, \sigma_{yy}, \sigma_{xy})$ or $(u_x, u_y, \varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy})$ as outputs of one network or multiple independent networks. As can be seen from Fig. 3, these choices affect the number of parameters of the network and how different quantities of interest are correlated. Equation (3) shows that the the feed-forward neural network imposes a special functional form to the network that may not

necessarily follow any cross-dependence between variables in the governing equations (1). Our data shows that using separate networks for each variable results in a far more effective strategy. Therefore, we propose to have variables $u_x, u_y, \sigma_{xx}, \sigma_{yy}, \sigma_{xy}$ defined as independent ANNs as our architecture of choice (see Fig. 4), i.e.

$$\begin{aligned} u_x(\mathbf{x}) &\approx \mathcal{N}_{u_x}(\mathbf{x}), \\ u_y(\mathbf{x}) &\approx \mathcal{N}_{u_y}(\mathbf{x}), \\ \sigma_{xx}(\mathbf{x}) &\approx \mathcal{N}_{\sigma_{xx}}(\mathbf{x}), \\ \sigma_{yy}(\mathbf{x}) &\approx \mathcal{N}_{\sigma_{yy}}(\mathbf{x}), \\ \sigma_{xy}(\mathbf{x}) &\approx \mathcal{N}_{\sigma_{xy}}(\mathbf{x}), \end{aligned} \tag{8}$$

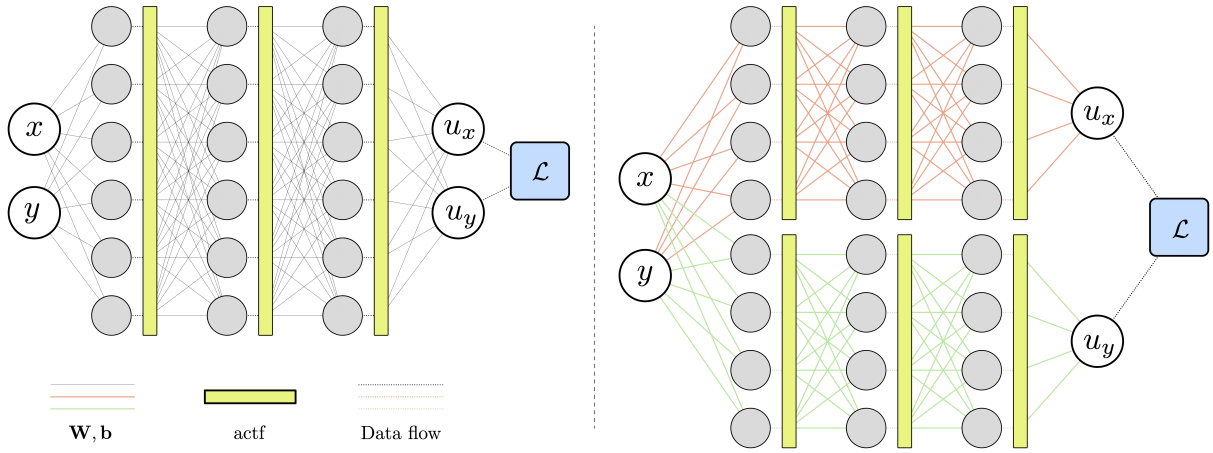

Figure 3: Potential PINN network choices, with u_x and u_y as outputs of a single network (left), or outputs of two independent networks with different parameters (right).

The cost function is defined as

$$\begin{aligned} \mathcal{L} = &|u_x - u_x^*| + |u_y - u_y^*| + |\sigma_{xx} - \sigma_{xx}^*| + |\sigma_{yy} - \sigma_{yy}^*| + |\sigma_{xy} - \sigma_{xy}^*| \\ &+ |\sigma_{xx,x} + \sigma_{xy,y} - f_x^*| + |\sigma_{xy,x} + \sigma_{yy,y} - f_y^*| \\ &+ |(\lambda + 2\mu)\varepsilon_{xx} + \lambda\varepsilon_{yy} - \sigma_{xx}| + |(\lambda + 2\mu)\varepsilon_{yy} + \lambda\varepsilon_{xx} - \sigma_{yy}| + |2\mu\varepsilon_{xy} - \sigma_{xy}|. \end{aligned} \tag{9}$$

The quantities with asterisks represent given data. We will train the networks so that their output values are as close as possible to the data, which may be real field data or, in this paper, synthetic data from the exact solution to the problem or the result of a high-fidelity simulation. The values without asterisk represent either direct outputs of the network (e.g., u_x or σ_{xx}; see Eq. (8)) or quantities obtained through automatic graph-based differentiation [35] of the network outputs (e.g., $\varepsilon_{xx} = u_{x,x}$). In Eq. (9), f_x^* and f_y^* represent data on the body forces obtained as $f_i^* = \sigma_{ij,j}^*$.

The different terms in the cost function represent measures of the error in the displacement and stress fields, the momentum balance, and the constitutive law. This cost function can be used for deep-learning-based solution of PDEs as well as for identification of the model parameters. For the solution of PDEs, λ and μ are treated as fixed numbers in the network. For parameter

identification, λ and μ are treated as network parameters that change during the training phase (see Fig. 4). In TensorFlow [30] this can be accomplished defining λ and μ as `Constant` (PDE solution) or `Variable` (parameter identification) objects, respectively. We set up the problem using the SciANN [40] framework, a high-level Keras [37] wrapper for physics-informed deep learning and scientific computations. Experimenting with all of the previously mentioned network choices can be easily done in SciANN with minimal coding.[1]

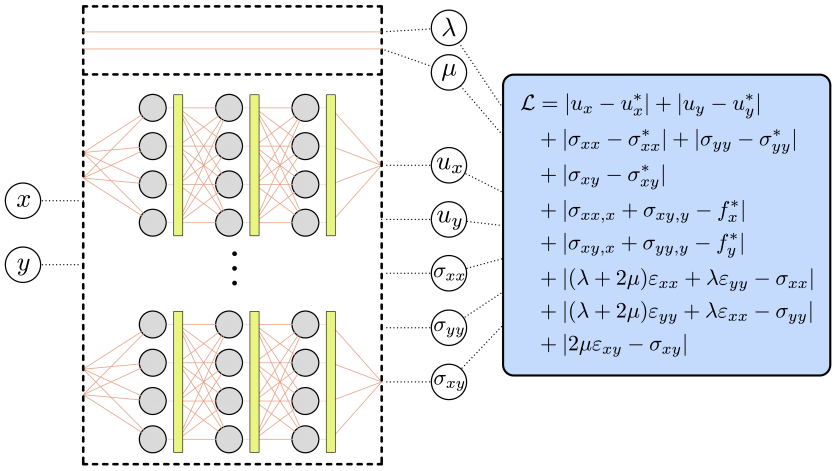

Figure 4: Network architecture of choice used in this study. We define five networks, one for each variable of interest, i.e., $u_x, u_y, \sigma_{xx}, \sigma_{yy}, \sigma_{xy}$. Each network has (x, y) as input features.

3.3. Identification of model parameters: PINN trained on the exact solution

Here, we use PINN to identify the model parameters λ and μ. Our data corresponds to the exact solution with parameter values $\lambda = 1$, $\mu = 0.5$ and $Q = 4$. Our default dataset consists of 100×100 sample points, uniformly distributed. We study how the accuracy and the efficiency of the identification process depend on the architecture and functional form of the network; the available data; and whether we use one or several independent networks for the different quantities of interest. To study the impact of the architecture and functional form of the ANN, we use 4 different networks with either 5 or 10 hidden layers, and either 20 or 50 neurons per layer; see Table 1. The role of the network functional form is studied comparing the performance of the two most widely used activation functions, i.e., $\tanh$ and ReLU, where $\text{ReLU}(x) = \max(0, x)$ [1].

Studying the impact of the available data on the identification process is crucial because we are interested in identifying the model parameters with as little data as possible. We undertake the analysis considering two scenarios:

(a) *Stress-complete data*: In this case, we have data at a set of points for the displacements and their first-order derivatives, that is, u_x^*, u_y^*, σ_{xx}^*, σ_{yy}^*, σ_{xy}^*. Because our cost function (9) involves also data that depends on the stress derivatives (f_x^* and f_y^*), this approach relies on an additional algorithmic procedure for differentiation of stresses. In this section we compute the stress derivatives using second-order central finite-difference approximations.

[1]The code for some of the examples solved here is available at: https://github.com/sciann/examples.

Table 1: Statistics of the networks of choice to perform PINN learning.

Network	Layers	Neurons	Number of Parameters	
			Independent Networks	Single Network
i	5	20	12336	1893
ii	5	50	72816	10713
iii	10	20	27036	3993
iv	10	50	162066	23463

(b) *Force-complete data*: In this scenario, we have data at a set of points for the displacements, their first derivatives and their second derivatives. The availability of the displacement second derivatives allows us to determine data for the body forces f_x^* and f_y^* using the momentum balance equation without resorting to any differentiation algorithm.

In Fig. 5 we compare the evolution of the cost function for stress-complete data (Fig. 5a) and force-complete data (Fig. 5b). Both figures show a comparison of the four network architectures that we study; see Table 1. We find that training on the force-complete data performs slightly better (lower loss) at a given epoch.

The result of convergence of model identification is shown in Fig. 6. The training converges to the true values of parameters, i.e., $\lambda = 1$ and $\mu = 1/2$, for all cases. We find that the optimization is very quick on the parameters while it takes far more epochs to fit the network on the field variables. Additionally, we observe that deeper networks produce less accurate parameters. We attribute the loss of accuracy as we increase the ANN complexity to over-fitting [1, 3]. Convergence of the individual terms in the loss function (9) is shown in Fig. 7 for Net-ii (see Table 1). We find that all terms in the loss, i.e., data-driven and physics-informed, show oscillations during the optimization. Therefore, no individual term is solely responsible for the oscillations in the total loss (Fig. 5).

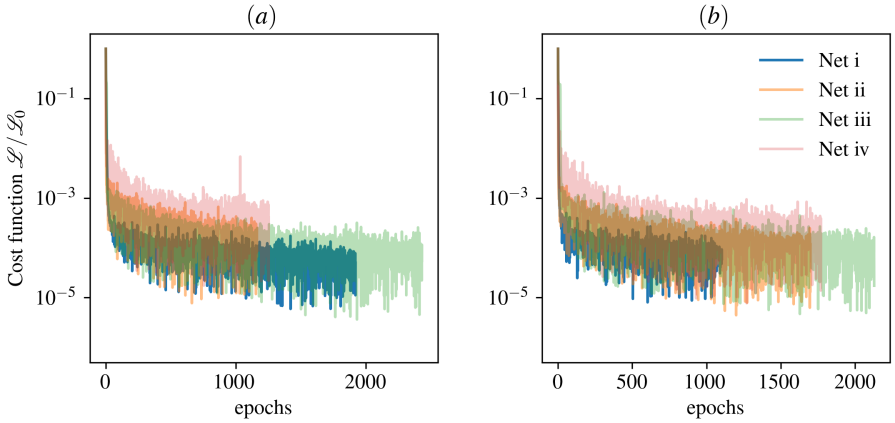

Figure 5: The result of training networks i, ii, iii, and iv on the analytical data set u_x, u_y, σ_{xx}, σ_{yy}, and σ_{xy}; (a) body forces are evaluated from central-difference differentiation of stress components, (b) body forces are also given analytically.

The impact of the ANN functional form can be examined comparing the data in Figs. 5b and 8a, which show the evolution of the cost function using the activation functions $\tanh$ and ReLU, respectively. The function ReLU has discontinuous derivatives, which explains its poor performance

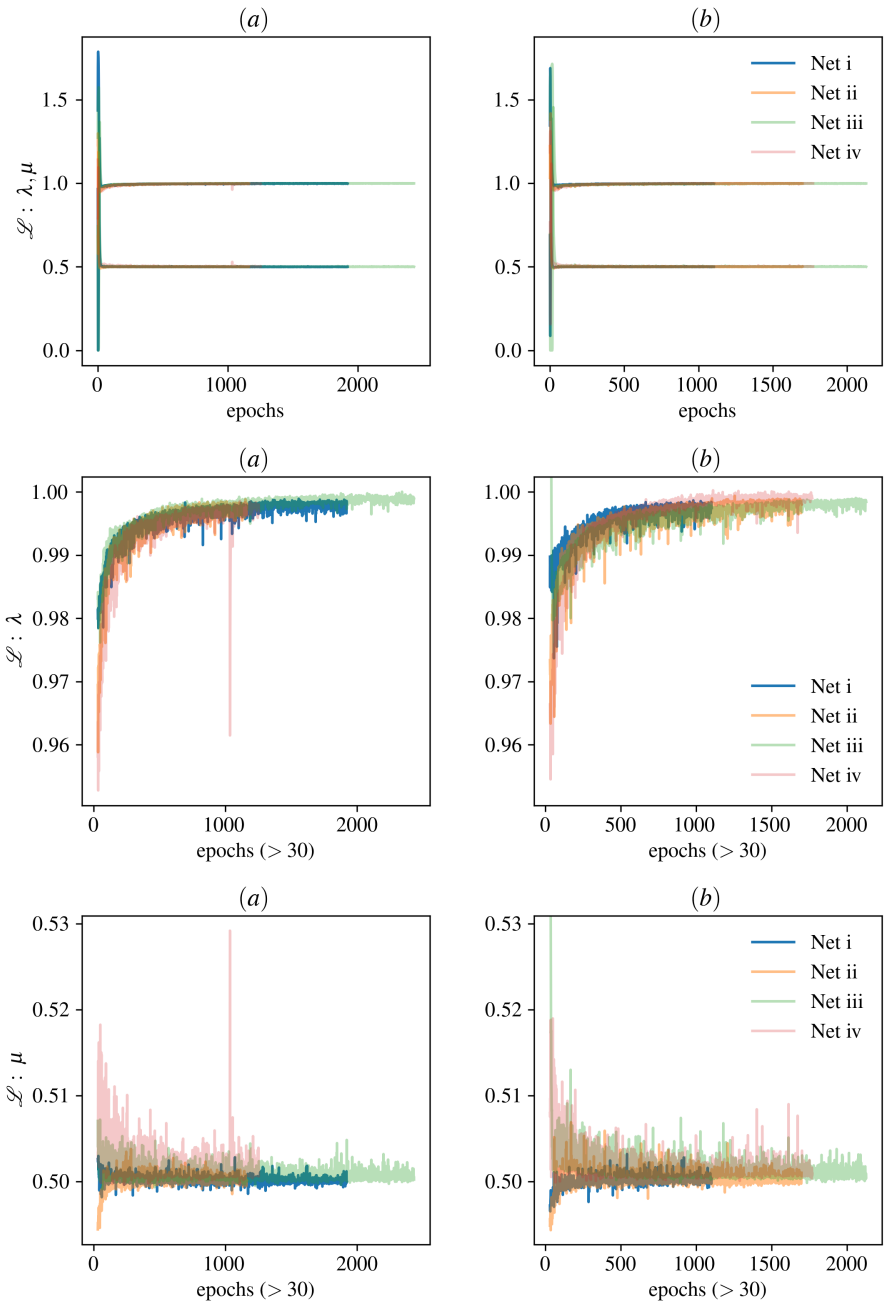

Figure 6: The result of identification for $\lambda = 1, \mu = 1/2$ for networks i, ii, iii, and iv on the analytical data set u_x, u_y, σ_{xx}, σ_{yy}, and σ_{xy}; (a) body forces are evaluated from central-difference differentiation of stress components, (b) body forces are also given analytically.

for physics-informed deep learning, whose effectiveness relies heavily on accurate evaluation of derivatives.

A comparison of Figs. 5b and 8b shows that using independent networks for displacements and stresses is more effective than using a single network. We find that the single network leads to less accurate elastic parameters because the cross-dependencies of the network outputs through the kinematic and constitutive relations may not be adequately represented by the $\tanh$ activation

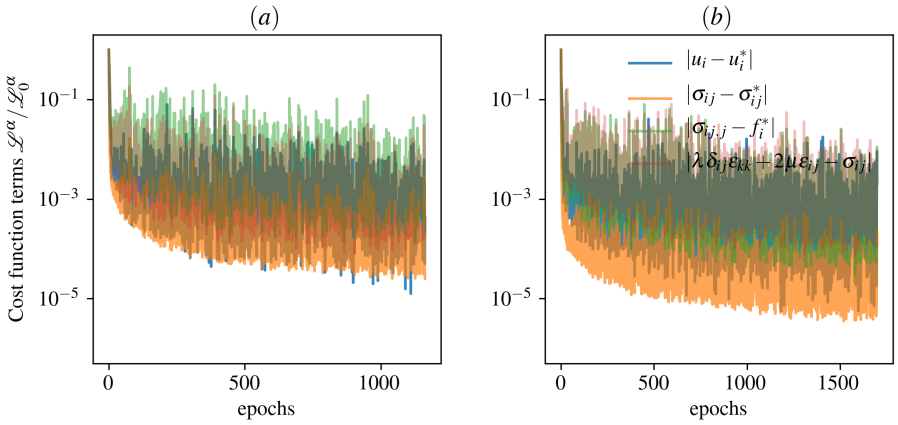

Figure 7: Individual terms of total loss (9) for network ii on the analytical data set u_x, u_y, σ_{xx}, σ_{yy}, and σ_{xy}; (a) body forces are evaluated from central-difference differentiation of stress components, (b) body forces are also given analytically.

function.

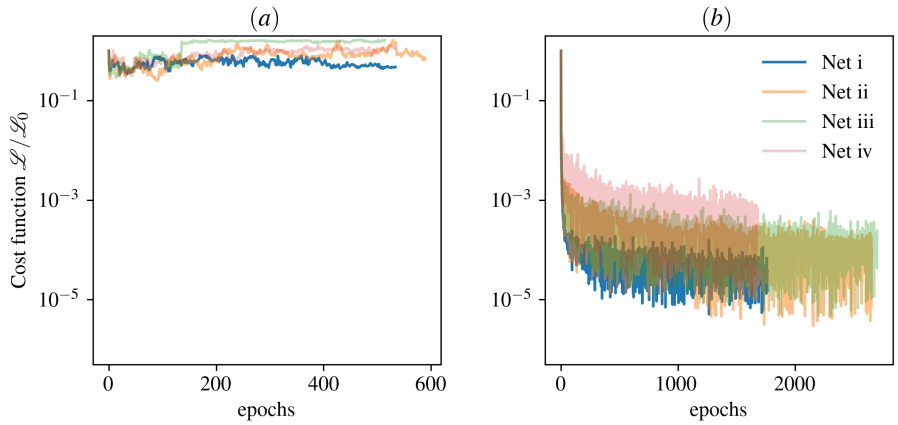

Figure 8: (a) ReLU activation function on the analytical data set u_x, u_y, σ_{xx}, σ_{yy}, σ_{xy}, f_x , and f_y. (b) Connected network.

Fig. 9 analyzes the effect of availability of data on the training. We computed the exact solution on four different uniform grids of size 10×10, 40×40, 160×160, and 640×640; and carried out the parameter identification process. We performed the comparison using force-complete data and a network with 10 layers and 20 neurons per layer (network iii). The training process found good approximations to the parameters for all cases, including that with only 10×10 points. The results show that fewer data points require many more epoch cycles, but the overall computational cost is far lower.

3.4. PINN models trained on the FEM solution

Here, we generate synthetic data from FEM solutions, and then perform the training. The domain is discretized with a mesh comprised of 40×40 elements. Four datasets are prepared using quadrilateral bilinear, biquadratic, bicubic, and biquartic Lagrange C^0 elements using the

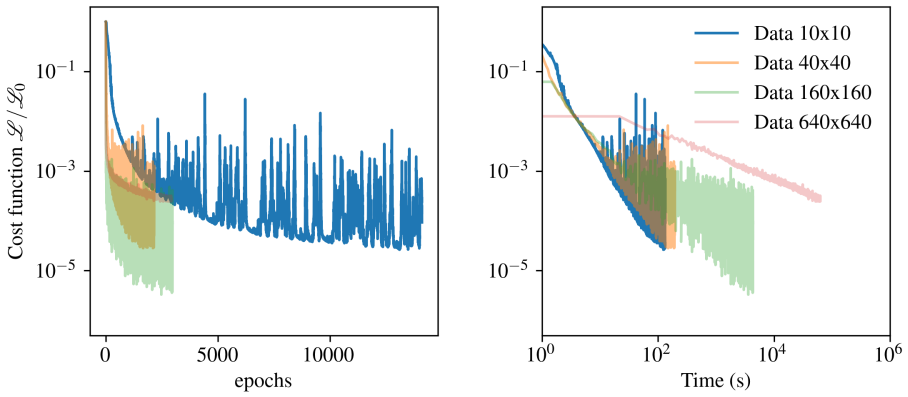

Figure 9: Training on different sizes of data. Parameters are all accurately identified however solution has different level of accuracy.

commercial FEM software COMSOL. We evaluate the FEM displacements, strains, stresses and stress derivatives at the center of each element. Then, we map the data to a 100×100 training grid using SciPy's griddata module with cubic interpolation. This step is performed as a data-augmentation procedure, which is a common practice in machine learning [1].

To analyze the importance of data satisfying the governing equations of the system, we focus our attention on network ii and we study cases with stress- and force-complete data. The results of training are presented in Fig. 10. As can be seen here, the bilinear element performs poorly on the learning and identification. The performance of training on the other elements is good, comparable to that using the analytical solution. Further analysis shows that this is indeed expected as FEM differentiation of bilinear elements provides a poor approximation of the body forces. The error in the body forces is shown in Fig. 11, which indicates a high error for bilinear elements. We conclude that the standard bilinear elements are not suitable for this problem to generate numerical data for deep learning. Fig. 10a2 confirms that pre-processing the data can remove the error that was present in the numerical solution with bilinear elements, and enable the optimization to successfully complete the identification.

3.5. PINN models trained on the IGA solution

Observing the lowest loss $\mathcal{L}$ on the analytical solution, we decided to study the influence of the global continuity of the numerical solution. We generated a C^3-continuous dataset using Isogeometric analysis [41]. We, therefore, analyze the system using C^3 IGA elements with again a grid of 40×40 dimension. The data are then mapped on to a grid of 100×100 and used to train the PINN models. The training results are shown in Fig. 12. The outputs are very similar to the high-order FEM datasets.

3.6. Identification using transfer learning

Here we explore the applicability of our PINN framework to transfer learning: a neural network that is pre-trained is used to perform identification on a new dataset. The expectation is that since the initial state of neural network is not randomly chosen anymore, training should converge faster to the solution and parameters of the data. This is crucial for many practical aspects including adaptation to new data for online search or purchase history [34] or in geosciences, where we can

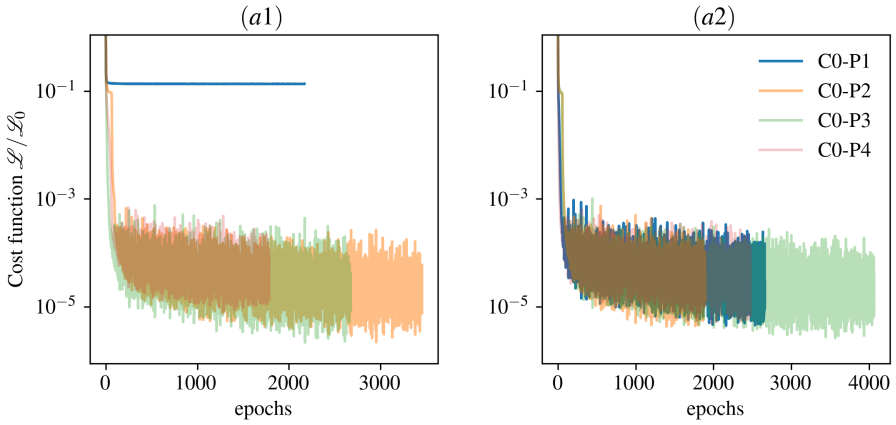

Figure 10: (a1) Training on the FEM dataset using u_x, u_y, σ_{xx}, σ_{yy}, σ_{xy}, f_x and f_y components. (a2) Training with body forces f_x and f_y evaluated from central-differentiation of stress components.

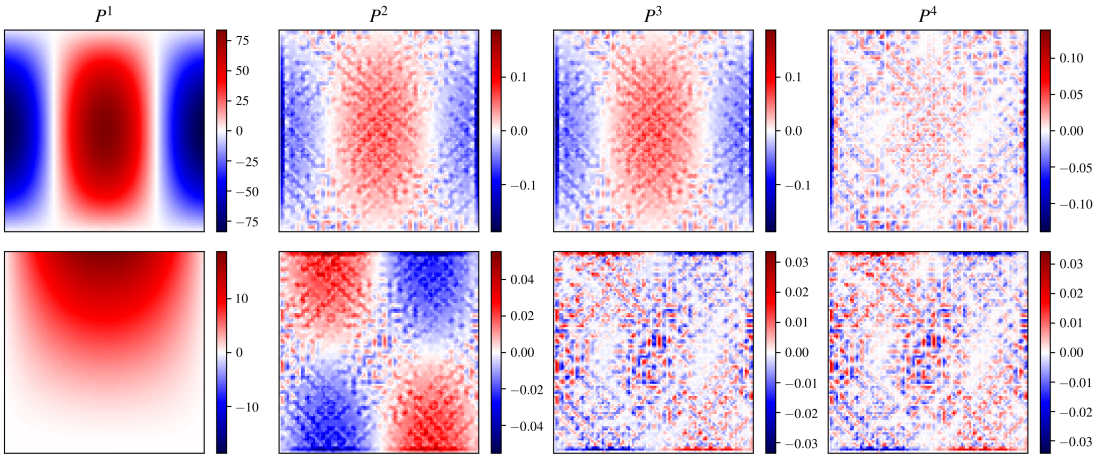

Figure 11: The error in bilinear, biquadratic, bicubic, and biquartic FEM data, that is evaluated as the difference between FEM evaluation of momentum relation, i.e., $\sigma_{ij,j}$ and true body forces f_i^* in x (top) and y (bottom) directions.

train a representative PINN in highly-instrumented regions and use them at other locations with limited observational datasets. To this end, we use the pre-trained model on Net-iii (Fig. 5), which was trained on a dataset with $\lambda = 1.0$ and $\mu = 0.5$ and then we explore how the loss evolves and the training converges when data is generated with different values of $\mu \in \{2.0, 1.5, 1.0, 0.1\}$.

In Fig. 13 we show the convergence of the model with different datasets. Note that the loss is normalized by the initial value $\mathcal{L}_0$ from the pre-trained network on $\mu = 0.5$ (Fig. 5). As can be seen here, re-training on new datasets costs only a few hundred epochs with a smaller initial value for the loss. This is pointing to the advantage of deep learning and PINN, where retraining on similar data is much less costly than classical methods that rely on forward simulations.

3.7. Application to sensitivity analysis

Performing sensitivity analysis is an expensive task when the analytical solution is not available, since it requires performing many forward numerical simulations. Alternatively, if we can

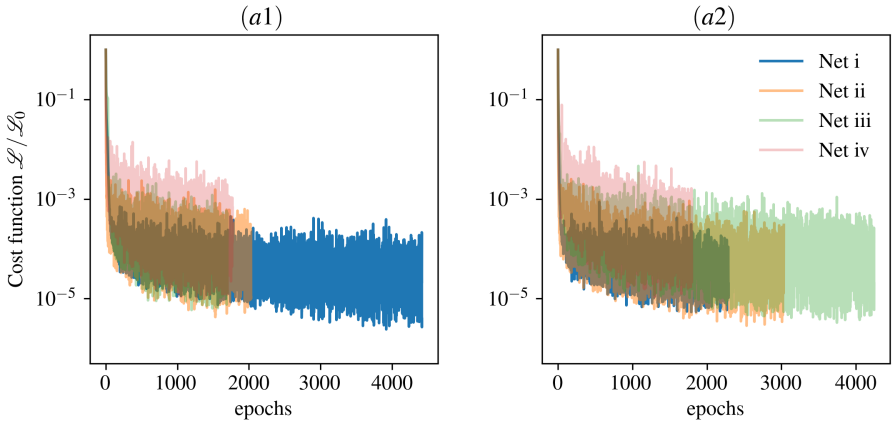

Figure 12: (a1) Training on the IGA dataset using u_x, u_y, σ_{xx}, σ_{yy}, σ_{xy}, f_x and f_y components. (a2) Learning with body forces f_x and f_y evaluated from centeral-differentiation of stress components.

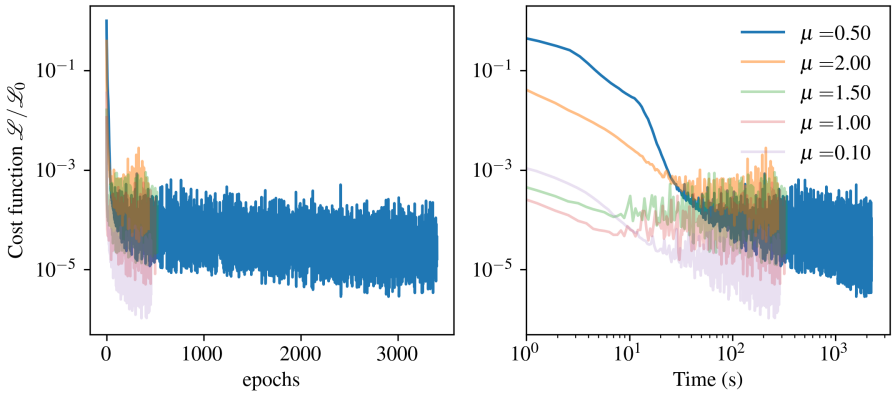

Figure 13: Identification of new dataset generated with different values of μ using a pre-trained neural network on $\mu = 0.5$. The re-training takes far less epochs to converge with an initial value for loss $\mathcal{L}$ much smaller.

construct a surrogate model to be a function of parameters of interest, then performing sensitivity analysis becomes tractable. However, construction of such a surrogate model is itself an expensive task within classical frameworks. Within PINN, however, this seems to be naturally possible. Let us suppose that the parameter of interest is shear modulus μ. Consider an ANN model with inputs as (x, y, μ) and outputs as $(u_x, u_y, \sigma_{xx}, \sigma_{yy}, \sigma_{xy})$. We can, therefore, use a similar framework to construct a model that is a function of μ in addition to the space variables. Again, PINN can constrain the model to adapt to the physics of interest and therefore there is less data needed to construct such a model.

Here, we explore if a PINN model trained on multiple datasets generated with various material parameters, i.e., different values of μ, can be used as a surrogate model to perform sensitivity analysis. The network in Fig. 4 is now slightly adapted to carry μ as an extra input (in addition to x, y). The training set is prepared based on $\lambda = 1$ and $\mu \in \{1/4, 2/3, 3/2, 4\}$. Note that there is no identification in this case, and therefore the parameters λ and μ are known at any given training data. The results of the analysis are shown in Fig. 14. For a wide range of values of $\mu \in (0, 9)$, the model performs very well in terms of displacements; it is less accurate, but still very useful, in

terms of stresses with a maximum error for near-incompressible conditions, $\mu \approx 0$.

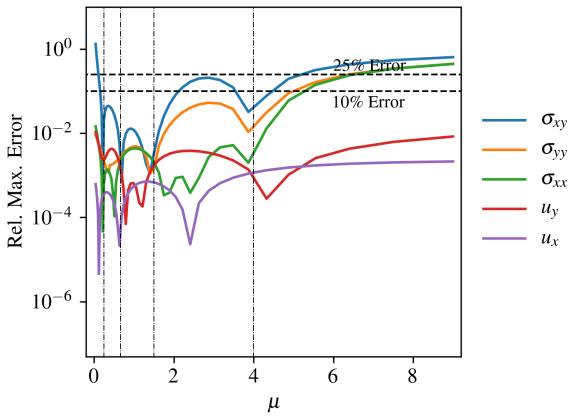

Figure 14: Application to sensitivity analysis: the model is trained on multiple datasets generated with different values of $\mu \in \{1/4, 2/3, 3/2, 4\}$ (highlighted in dot-dashed lines). The model is then tested on a continuous range of values for $\mu \in (0, 9)$. The error is defined as $|\circ - \circ^*|/|\circ^*|$ at the point where $\circ^*$ is maximum.

4. Conclusions

We study the application of a new class of deep learning, known as Physics-Informed Neural Networks (PINN), for solution and discovery in solid mechanics. In this work, we formulate and apply the framework to a linear elastostatics problem. We study the sensitivity of the proposed framework to noise in data coming from different numerical techniques. We find that the optimizer performs much better on data from high-order classical finite elements, or with methods with enhanced continuity such as Isogeometric Analysis. We analyze the impact of the size and depth of the network, and the size of the dataset from uniform sampling of the numerical solutionan aspect that is important in practice given the cost of a dense monitoring network. We find that the proposed PINN approach is able to converge to the solution and identify the parameters quite efficiently with as little as 100 data points.

We also explore transfer learning, that is, the use a pre-trained neural network to perform training on new datasets with different parameters. We find that training converges much faster when this is done. Lastly, we study the applicability of the model as a surrogate model for sensitivity analysis. To this end, we introduce shear modulus μ as an input variable to the network. When training only on four values of μ, we find that the network predicts the solution quite accurately on a wide range of values for μ, a feature that is indicative of the robustness of the approach.

Despite the success exhibited by the PINN approach, we have found that it faces challenges when dealing with problems with discontinuous solutions. The network architecture is less accurate on problems with localized high gradients as a result of discontinuities in the material properties or boundary conditions. We find that, in those cases, the results are artificially diffuse where they should be sharp. We speculate that the underlying reason for this behavior is the particular architecture of the network, where the input variables are only the spatial dimensions (x and y), rendering the network unable to produce the required variability needed for gradient-based optimization that would capture solutions with high gradients. Addressing this extension is an exciting avenue for future work in machine-learning applications to solid mechanics.

References

[1] C. M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag, Berlin, Heidelberg, 2006. URL: https://www.springer.com/gp/book/9780387310732. doi:https://dl.acm.org/doi/book/10.5555/1162264.

[2] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444. URL: https://doi.org/10.1038/nature14539. doi:10.1038/nature14539.

[3] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, 2016. URL: https://www.deeplearningbook.org. doi:https://dl.acm.org/doi/book/10.5555/3086952.

[4] C. E. Yoon, O. O'Reilly, K. J. Bergen, G. C. Beroza, Earthquake detection through computationally efficient similarity search, Science Advances 1 (2015) e1501057. URL: http://advances.sciencemag.org/lookup/doi/10.1126/sciadv.1501057. doi:10.1126/sciadv.1501057.

[5] K. J. Bergen, P. A. Johnson, M. V. de Hoop, G. C. Beroza, Machine learning for data-driven discovery in solid earth geoscience, Science 363 (2019). URL: https://science.sciencemag.org/content/363/6433/eaau0323. doi:10.1126/science.aau0323. arXiv:https://science.sciencemag.org/content/363/6433/eaau0323.full.pdf.

[6] P. M. DeVries, F. Viégas, M. Wattenberg, B. J. Meade, Deep learning of aftershock patterns following large earthquakes, Nature 560 (2018) 632–634. URL: http://dx.doi.org/10.1038/s41586-018-0438-y. doi:10.1038/s41586-018-0438-y.

[7] Q. Kong, D. T. Trugman, Z. E. Ross, M. J. Bianco, B. J. Meade, P. Gerstoft, Machine learning in seismology: turning data into insights, Seismological Research Letters 90 (2018) 3–14. doi:10.1785/0220180259.

[8] C. X. Ren, O. Dorostkar, B. Rouet-Leduc, C. Hulbert, D. Strebel, R. A. Guyer, P. A. Johnson, J. Carmeliet, Machine learning reveals the state of intermittent frictional dynamics in a sheared granular fault, Geophysical Research Letters 46 (2019) 7395–7403. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019GL082706. doi:10.1029/2019GL082706. arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019GL0827

[9] G. Pilania, C. Wang, X. Jiang, S. Rajasekaran, R. Ramprasad, Accelerating materials property predictions using machine learning, Scientific Reports 3 (2013) 1–6. doi:10.1038/srep02810.

[10] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, A. Walsh, Machine learning for molecular and materials science, Nature 559 (2018) 547–555. URL: http://dx.doi.org/10.1038/s41586-018-0337-2. doi:10.1038/s41586-018-0337-2.

[11] Z. Shi, E. Tsymbalov, M. Dao, S. Suresh, A. Shapeev, J. Li, Deep elastic strain engineering of bandgap through machine learning, Proceedings of the National Academy of Sciences 116 (2019) 4117–4122. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1818555116. doi:10.1073/pnas.1818555116.

[12] S. L. Brunton, J. N. Kutz, Methods for data-driven multiscale model discovery for materials, Journal of Physics: Materials 2 (2019) 044002. URL: https://doi.org/10.1088%2F2515-7639%2Fab291e. doi:10.1088/2515-7639/ab291e.

[13] M. P. Brenner, J. D. Eldredge, J. B. Freund, Perspective on machine learning for advancing fluid mechanics, Physical Review Fluids 4 (2019) 100501. URL: https://link.aps.org/doi/10.1103/PhysRevFluids.4.100501. doi:10.1103/PhysRevFluids.4.100501.

[14] S. L. Brunton, B. R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annual Review of Fluid Mechanics 52 (2020) 477–508. URL: https://doi.org/10.1146/annurev-fluid-010719-060214. doi:10.1146/annurev-fluid-010719-060214.

[15] M. W. Libbrecht, W. S. Noble, Machine learning applications in genetics and genomics, Nature Reviews Genetics 16 (2015) 321–332. URL: http://dx.doi.org/10.1038/nrg3920. doi:10.1038/nrg3920.

[16] M. H. Rafiei, H. Adeli, A novel machine learning-based algorithm to detect damage in high-rise building structures, Structural Design of Tall and Special Buildings 26 (2017) 1–11. doi:10.1002/tal.1400.

[17] D. Sen, A. Aghazadeh, A. Mousavi, S. Nagarajaiah, R. Baraniuk, A. Dabak, Data-driven semi-supervised and supervised learning algorithms for health monitoring of pipes, Mechanical Systems and Signal Processing 131 (2019) 524–537. URL: https://doi.org/10.1016/j.ymssp.2019.06.003. doi:10.1016/j.ymssp.2019.06.003.

[18] J. Ghaboussi, D. Sidarta, New nested adaptive neural networks (NANN) for constitutive modeling, Computers and Geotechnics 22 (1998) 29–52.

[19] S. R. Kalidindi, S. R. Niezgoda, A. A. Salem, Microstructure informatics using higher-order statistics and efficient data-mining protocols, JOM 63 (2011) 34–41. URL: https://doi.org/10.1007/s11837-011-0057-7. doi:10.1007/s11837-011-0057-7.

[20] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, M. A. Bessa, Deep learning predicts path-dependent plasticity, Proceedings of the National Academy of Sciences 116 (2019) 26414–26420. URL: https://www.pnas.org/content/116/52/26414. doi:10.1073/pnas.1911815116. arXiv:https://www.pnas.org/content/116/52/26414.full.pdf.

[21] S. Rudy, A. Alla, S. L. Brunton, J. N. Kutz, Data-driven identification of parametric partial differential equations, SIAM Journal on Applied Dynamical Systems 18 (2019) 643–660. URL: https://epubs.siam.org/doi/abs/10.1137/18M1191944. doi:10.1137/18M1191944.

[22] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707. URL: https://doi.org/10.1016/j.jcp.2018.10.045. doi:10.1016/j.jcp.2018.10.045.

[23] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences 115 (2018) 8505–8510. URL: https://www.pnas.org/content/115/34/8505. doi:10.1073/pnas.1718942115.

[24] Y. Bar-Sinai, S. Hoyer, J. Hickey, M. P. Brenner, Learning data-driven discretizations for partial differential equations, Proceedings of the National Academy of Sciences 116 (2019) 15344–15349. URL: https://www.pnas.org/content/116/31/15344. doi:10.1073/pnas.1814058116. arXiv:https://www.pnas.org/content/116/31/15344.full.pdf.

[25] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, Journal of Computational Physics 394 (2019) 56–81. URL: https://www.sciencedirect.com/science/article/pii/S0021999119303559. doi:10.1016/j.jcp.2019.05.024.

[26] A. J. Meade, A. A. Fernandez, The numerical solution of linear ordinary differential equations by feed-forward neural networks, Mathematical and Computer Modelling 19 (1994) 1–25. URL: https://www.sciencedirect.com/science/article/pii/0895717794900957. doi:10.1016/0895-7177(94)90095-7.

[27] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (1998) 987–1000. URL: https://ieeexplore.ieee.org/document/712178. doi:10.1109/72.712178.

[28] I. E. Lagaris, A. C. Likas, D. G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Transactions on Neural Networks 11 (2000) 1041–1049. URL: https://ieeexplore.ieee.org/document/870037. doi:10.1109/72.870037.

[29] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, Theano: a CPU and GPU math expression compiler, in: Proceedings of the Python for Scientific Computing Conference (SciPy), volume 4, Austin, TX, 2010.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, Savannah,

GA, 2016, pp. 265–283. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[31] S. Lange, T. Gabel, M. Riedmiller, Reinforcement learning, volume 12 of *Adaptation, Learning, and Optimization*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: http://link.springer.com/10.1007/978-3-642-27645-3. doi:10.1007/978-3-642-27645-3.

[32] T. Hughes, J. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Computer Methods in Applied Mechanics and Engineering 194 (2005) 4135–4195. URL: http://www.sciencedirect.com/science/article/pii/S0045782504005171. doi:https://doi.org/10.1016/j.cma.2004.10.008.

[33] J. A. Cottrell, T. J. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Integration of CAD and FEA, John Wiley & Sons, 2009. doi:https://dl.acm.org/doi/book/10.5555/1816404.

[34] M. E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: A survey, Journal of Machine Learning Research 10 (2009) 1633–1685. URL: http://www.jmlr.org/papers/v10/taylor09a.html. doi:https://dl.acm.org/doi/10.5555/1577069.1755839.

[35] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: a survey, The Journal of Machine Learning Research 18 (2017) 5595–5637. URL: https://dl.acm.org/doi/abs/10.5555/3122009.3242010.

[36] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems (2015). arXiv:1512.01274.

[37] F. Chollet, et al., Keras, 2015.

[38] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014). arXiv:1412.6980.

[39] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (2011) 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[40] E. Haghighat, R. Juanes, SciANN: A Keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks, https://sciann.com, 2019. URL: https://github.com/sciann/sciann.git.

[41] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, T. W. Sederberg, Isogeometric analysis using T-splines, Computer Methods in Applied Mechanics and Engineering 199 (2010) 229–263. URL: https://www.sciencedirect.com/science/article/pii/S0045782509000875. doi:10.1016/j.cma.2009.02.036.