Physics-informed PointNet: On how many irregular geometries can it solve an inverse problem simultaneously? Application to linear elasticity

Ali Kashefi a*, Leonidas J. Guibas b, Tapan Mukerji c

aDepartment of Civil and Environmental Engineering, Stanford University, Stanford, CA, 94305 USA
bDepartment of Computer Science, Stanford University, Stanford, CA, 94305 USA
cDepartment of Energy Science and Engineering, Stanford University, Stanford, CA, 94305 USA

*Corresponding author: Ali Kashefi, kashefi@stanford.edu

Abstract

Regular physics-informed neural networks (PINNs) predict the solution of partial differential equations using sparse labeled data but only over a single domain. On the other hand, fully supervised learning models are first trained usually over a few thousand domains with known solutions (i.e., labeled data) and then predict the solution over a few hundred unseen domains. Physics-informed PointNet (PIPN) is primarily designed to fill this gap between PINNs (as weakly supervised learning models) and fully supervised learning models. In this article, we demonstrate that PIPN predicts the solution of desired partial differential equations over a few hundred domains simultaneously, while it only uses sparse labeled data. This framework benefits fast geometric designs in the industry when only sparse labeled data are available. Particularly, we show that PIPN predicts the solution of a plane stress problem over more than 500 domains with different geometries, simultaneously. Moreover, we pioneer implementing the concept of remarkable batch size (i.e., the number of geometries fed into PIPN at each sub-epoch) into PIPN. Specifically, we try batch sizes of 7, 14, 19, 28, 38, 76, and 133. Additionally, the effect of the PIPN size, symmetric function in the PIPN architecture, and static and dynamic weights for the component of the sparse labeled data in the loss function are investigated.

Keywords: Physics-informed deep learning; PointNet; Irregular geometries; Linear elasticity; Inverse problems

Highlights

1. Similar to supervised deep learning, we test big data (i.e., number of geometry) and relatively large batch sizes (i.e., number of geometries per epoch) for the framework of physics-informed neural networks for the first time.
2. We specifically use PIPN as an advanced version of physics-informed neural networks.
3. Using PIPN, an inverse problem of linear elasticity is solved over 532 irregular geometries simultaneously.
4. Batch sizes (i.e., number of geometries per epoch) of 7, 14, 19, 28, 38, 76, and 133 are tested.

1 Introduction and motivation

Physics-informed Neural Networks (PINNs), introduced by Raissi, et al. [42] in 2019, are recognized as a promising tool for solving inverse problems in a variety of scientific and industrial fields such as solid mechanics [15, 43, 21, 47, 49, 54, 12, 8, 52, 3, 45, 19, 41, 11], incompressible and compressible flows [22, 25, 32, 18, 17, 42, 37, 40, 36, 6], chemistry [51, 20], heat transfer [25, 50, 7], flow in porous media [26, 2, 57], etc. The main idea of PINNs for solving inverse problems can be briefly explained as follows. Given sparse observations of a field of interest as well as the partial differential equations (PDEs) governing the physics of the field, train “a neural network” such that its predictions (outputs) minimize the residuals of the PDEs as well as the distance between the predictions and sparse observations at sensor locations, in a certain norm (such as $L^2$ norm or other norms). Other partial information such as boundary or initial conditions may be included in this minimization problem.
Our focus, here, is on the term “a neural network”. We believe that the choice of the neural network significantly affects the ability and capacity of a PINN configuration. A common choice is a fully connected neural network (see e.g., see Fig. 1 in Ref. [43], Fig. 3 in Ref. [13], Fig. 2 in Ref. [55], Fig. 1 in Ref. [32], Fig. 3 in Ref. [58], Fig 1 in Ref. [56], Fig. 1 in Ref. [33], Fig. 4 in Ref. [31], Fig. 1 in Ref. [10], etc.). An immediate consequence of this choice is that a PINN with a fully connected neural network is only able to predict the solution of an inverse problem on a single geometry. Hence, for any computational domain with a new geometry, one needs to train a PINN from scratch. This scenario necessitates high computational expenses, specifically when the goal is the investigation of a wide range of geometric parameters for optimizing an industrial design. This issue has been first addressed by Gao, et al. [13] in 2021 and later by Kashefi and Mukerji [25] in 2022.

To resolve this issue, Gao, et al. [13] first proposed the PhyGeoNet [13] framework and later Kashefi and Mukerji [25] introduced the PIPN [25] framework. PIPN [25] also successfully overcame the shortcomings of PhyGeoNet [13]. We later compare PIPN [25] versus PhyGeoNet [13], but for now, let us focus on the main theme of our article. What was one of the main motivations for introducing PIPN? To answer this question, let us discuss PINNs from another perspective. PINNs can be categorized as weakly supervised learning frameworks in the sense that the associated neural networks predict the solution on a computational domain using some sparse labeled data, which are available on that specific domain. On the other hand, neural networks used in supervised learning methods for computational mechanics are first trained on a large number of computational domains with different geometries, where the solutions are known, and then they predict the solution on new computational domains with new geometries. Supervised learning methods are commonly trained on a few hundred to a few thousand geometries (e.g., 2595 in Ref. [27], 1505 in Ref. [48], 880 in Ref. [46], 1600 in Ref. [9], etc.). One of the goals of introducing PIPN was to fill the gap between the weakly-supervised learning and supervised learning models in terms of the number of geometries for which we can predict the solution simultaneously (see Fig. 1). Now, the question is given the current capacity of available commercial graphics processing units (GPUs) to the public, how many computational domains (with different geometries) is PIPN able to predict the solution of an inverse problem on, simultaneously?

The answer to this question depends on different parameters. But definitely one of them is the level of complexity of a problem, which has a direct relevance to its governing PDEs. When Kashefi and Mukerji [25] introduced PIPN, they [25] showed its applications for incompressible flows and natural convection, where the governing PDEs are nonlinear (see Eqs. 1–3 of Ref. [25]). Due to the nonlinearity of the problem, they had to select high resolutions (number of inquiry points) per computational domain. Specifically for the natural convection problem (see Sect. 4.2 of Ref. [25]), they run PIPN on 108 geometries, while each geometry had a resolution of 5000 inquiry points. They [25] further showed the effect of resolution on the accuracy of the PIPN prediction (see Fig. 18 of Ref. [25]). Because of the current limitations on GPU memories, they were not able to explore a “big” data set comparable to those used in supervised learning. One way to reduce the complexity level is to select a set of linear PDEs to examine the PIPN capacity. A reasonable choice is the equations of two-dimensional linear elasticity problems, which have a wide range of applications in the industry. Hence, to examine the capacity of PIPN in terms of simultaneous handling of the maximum possible number of geometries, we focus on the two-dimensional linear elasticity problems.

An important hyperparameter that needs to be tuned for training neural networks in supervised learning of computational mechanics is “batch size”. The concept of “batch size” is defined under the category of mini-batch gradient descent [14], where the training data set is split into smaller data sets as mini-batches. The size of each mini-batch is then called “batch size”. The batch size plays a critical role in the training convergence as well as the ability of a trained neural network to generalize predictions [23, 28, 4, 34]. In supervised deep learning of quantities of interest as a function of geometric features of domains, the term batch size refers to the number of domains, with different geometries, that are fed to a neural network at each epoch. In regular PINNs, however, the term batch size refers to the number of inquiry points belonging to a sub-domain of a single domain that is fed to a neural network at each epoch. In PIPN, Kashefi and Mukerji [25] used the term batch size to address the number of domains at each epoch, similar to the scenario for supervised learning. Nevertheless, the maximum batch size was 13 as reported in their research article [25], again due to the GPU memory limitations and nonlinearity of their considered PDEs. In this article, we investigate a wide range of batch sizes and their influences on the training convergence and generalizability of PIPN with its applications to a linear elasticity problem. Lastly, we study another important hyperparameter, which is the network size. In the framework of regular PINNs, where fully connected layers are used, the network size is tuned by changing the number of layers and a number of neurons in each layer. Contrarily to regular PINNs, PIPN has a more complicated architecture, and its network size can practically vary in different ways. In this work, we define a global scalar variable to control the network size of PIPN. Afterward, we study the performance of the PIPN framework as a function of this global variable.

For the sake of completeness and also convincing the potential audiences why we go with PIPN [25] rather than PhyGeoNet [13] in this article, we compare PIPN [25] with PhyGeoNet [13]. In PhyGeoNet [13], a convolutional neural network (CNN) is used as the “neural network” in a PINN, instead of a fully connected neural network. CNNs are able to extract geometric features of input computational domains and represent these features in a hidden space. Hence, all the parameters of CNNs (e.g., weights and bias) become a function of the geometric features. As a result, PhyGeoNet [13] is able to solve an inverse problem on multiple sets of computational domains. Notwithstanding these successes, PhyGeoNet [13] and its later version [44] come with several shortcomings. These
Physics-Informed Neural Networks (PINNs) are fed into the model at each sub-epoch. Prediction on a single geometry formulated as follows: given the temperature field of the domains \( \Phi = \{V_i\}_{i=1}^n \) and the \\
material of the domain \( V \), respectively. We assume that the material of the domain \( V \) is placed under a thermal conductivity loading. The static linear elasticity equations governing the displacement fields in the domain \( V \) are given by

\[
\begin{align*}
-\frac{\partial}{\partial x} \left( \frac{E}{1 - \nu^2} \frac{\partial u}{\partial x} + \frac{E\nu}{1 - \nu^2} \frac{\partial u}{\partial y} \right) - \frac{\partial}{\partial y} \left( \frac{E}{2(1 + \nu)} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right) &= - \frac{E\alpha}{1 - \nu^2} \frac{\partial T}{\partial x}, \\
-\frac{\partial}{\partial y} \left( \frac{E\nu}{1 - \nu^2} \frac{\partial u}{\partial x} + \frac{E}{1 - \nu^2} \frac{\partial v}{\partial y} \right) - \frac{\partial}{\partial x} \left( \frac{E}{2(1 + \nu)} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right) &= - \frac{E\alpha}{1 - \nu^2} \frac{\partial T}{\partial y},
\end{align*}
\]

where \( T \) indicates the temperature variable. Displacements in the \( x \) and \( y \) directions are shown by \( u \) and \( v \), respectively. \( \alpha \) denotes the thermal expansion coefficient. \( E \) is the elastic Young’s modulus and \( \nu \) is the Poisson ratio. We assume that the material of the domain \( V \) remains in the elastic regime under thermal loading.

Mathematically, our goal is to solve an inverse problem using PIPN on a set of irregular domains \( \Phi = \{V_i\}_{i=1}^n \), formulated as follows: given the temperature field of the domains \( \Phi = \{V_i\}_{i=1}^n \) and a set of sparse observations

Figure 1: A conceptual comparison between regular physics-informed neural networks (PINNs), physics-informed PointNet (PIPN), and fully supervised deep learning algorithms for computational mechanics in terms of the number of geometries and the batch size definition.

shortcomings have been addressed in detail in Sect. 1 of Ref. [25]. Here, we summarize them. First, PhyGeoNet [13] uses a finite difference discretization method for computing the loss function of desired PDEs rather than using the automatic differentiation technology [1]. In this way, PhyGeoNet [13] is limited to the accuracy order of the chosen finite difference scheme and faces challenges for treating near boundaries of computational domains in the case of using high-order finite difference methods. Second, PhyGeoNet [13] is not able to handle irregular “non-parameterizable” geometries. Third, even on irregular parameterizable geometries, PhyGeoNet [13] is strictly limited to handling domains with up to only five \( C_0 \) continuous boundaries.

To obviate the limitations of PhyGeoNet [13], Kashefi and Mukerji [25] introduced physics-informed PointNet (PIPN). In PIPN, PointNet [38] carries out the role of the “neural network” in PINNs. Similar to CNNs, PointNet [38] is able to also encode the geometric features of input computational domains; however, using a different mathematical methodology. Details of the PointNet architecture can be found in Refs. [38, 27, 25, 24]. We also briefly review the PointNet structure in Sect. 2.2 of this research paper. The advantages of PIPN compared to PhyGeoNet have been explained in detail by Kashefi and Mukerji [25]. We list them here for a quick review. First and most important, PIPN [25] can handle non-parameterizable irregular geometries without any restriction. Second, PIPN [25] uses automatic differentiation technology [1] and thus the spatial derivative of output components with respect to the corresponding input component can be conveniently programmed and computed over entire irregular geometries in a data set (with no need for a finite difference/element discretization). Third, in the PIPN framework, interior and boundary points of irregular geometries are explicitly identified, and it allows twofold benefits: first, a smooth representation of boundaries (e.g., an airfoil surface), and second, explicit identification of their corresponding components in the loss function (with no need for implicit labeling of boundary points versus interior points and an artificial mixing up near boundaries, especially with sharp corners).

The rest of this research paper is structured as follows. We mathematically formulate the linear elasticity problem in Sect. 2.1. An illustration of PIPN from computer science and applied mathematics perspectives are given in Sect. 2.2. Details about generating data for validation of the PIPN framework are described in Sect. 2.3. We elaborate on the computational setup for training PIPN in Sect. 2.4. A general analysis of the PIPN performance as well as the effect of batch size and neural network size is discussed in Sect. 3. Concluding remarks are listed in Sect. 4.

2 Problem statement and methodologies

2.1 Problem formulations

In this article, we focus on elastic materials with the constitutive model of Hooke’s law, relating the Cauchy stress tensor to the infinitesimal strain tensor and we restrict our studies to isotropic materials under plane stress assumptions. Specifically, we consider a two-dimensional domain \( V \) having a cavity space characterized by various shapes. The domain \( V \) is placed under a thermal conductivity loading. The static linear elasticity equations governing the displacement fields in the domain \( V \) are given by

\[
\begin{align*}
-\frac{\partial}{\partial x} \left( \frac{E}{1 - \nu^2} \frac{\partial u}{\partial x} + \frac{E\nu}{1 - \nu^2} \frac{\partial u}{\partial y} \right) - \frac{\partial}{\partial y} \left( \frac{E}{2(1 + \nu)} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right) &= - \frac{E\alpha}{1 - \nu^2} \frac{\partial T}{\partial x}, \\
-\frac{\partial}{\partial y} \left( \frac{E\nu}{1 - \nu^2} \frac{\partial u}{\partial x} + \frac{E}{1 - \nu^2} \frac{\partial v}{\partial y} \right) - \frac{\partial}{\partial x} \left( \frac{E}{2(1 + \nu)} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right) &= - \frac{E\alpha}{1 - \nu^2} \frac{\partial T}{\partial y},
\end{align*}
\]

where \( T \) indicates the temperature variable. Displacements in the \( x \) and \( y \) directions are shown by \( u \) and \( v \), respectively. \( \alpha \) denotes the thermal expansion coefficient. \( E \) is the elastic Young’s modulus and \( \nu \) is the Poisson ratio. We assume that the material of the domain \( V \) remains in the elastic regime under thermal loading. Mathematically, our goal is to solve an inverse problem using PIPN on a set of irregular domains \( \Phi = \{V_i\}_{i=1}^n \), formulated as follows: given the temperature field of the domains \( \Phi = \{V_i\}_{i=1}^n \) and a set of sparse observations
of the displacement fields of the domains of the set \( \Phi = \{ V_i \}_{i=1}^m \), find the full displacement solutions for all the domains of the set \( \Phi = \{ V_i \}_{i=1}^m \). We explicitly illustrate geometric variations of the set \( \Phi = \{ V_i \}_{i=1}^m \) in Sect. 2.3.

### 2.2 Physics-informed PointNet (PIPN)

As we discussed in Sect. 1, the idea of PIPN was first proposed by Kashefi and Mulerji [25] in 2022. Here, we briefly review the PIPN methodology and adjust it for the governing equations of our interest (Eqs. 1–2).

PIPN is built based on the combination of two critical pieces: PointNet [38] and a physics-based loss function. In simple words, the PIPN mechanism can be explained in two steps. In the first step, PointNet [38] makes the predicted outputs as a function of geometric features of each \( V_i \in \Phi \). As the second step, the PIPN loss function is computed by taking these outputs and manipulating them using automatic differentiation [1] to build up the PIPN loss function in the following section.

We define the PIPN architecture shown in Fig. 2. We further explain the role of shared MLPs and the symmetric functions in the following section.

### 2.2.1 Architecture

The PIPN architecture is illustrated in Fig. 2. In PIPN, each \( V_i \) is represented by a point-cloud \( \mathcal{X}_i \) with \( N \) points. \( \mathcal{X}_i \) is defined as \( \mathcal{X}_i = \{ x_j \in \mathbb{R}^d \}_{j=1}^N \), where \( d \) is the spatial dimension and we set \( d = 2 \) in this research study. Thus, each \( x_j \) has two components \( x_j \) and \( y_j \) as the spatial coordinates. PIPN maps \( \mathcal{X}_i \) to \( \mathcal{Y}_i \) via a function \( f \), where \( \mathcal{Y}_i \) is the prediction of the PDE (Eqs. 1–2) solutions. \( \mathcal{Y}_i \) is defined as \( \mathcal{Y}_i = \{ y_j \in \mathbb{R}^{n_{\text{pred}}} \}_{j=1}^N \), where \( n_{\text{pred}} \) indicates the number of fields in the solution. Here, we set \( n_{\text{pred}} = 2 \) as the displacement fields \( u \) and \( v \) are the unknowns. Hence, each \( y_j \) has two components, \( u_j \) and \( v_j \) as the network outputs. Mathematically, it can be written as

\[
(u_j, v_j) = f((x_j, y_j), g(\mathcal{X}_i)) \forall (x_j, y_j) \in \mathcal{X}_i \text{ and } \forall (u_j, v_j) \in \mathcal{Y}_i \text{ with } 1 \leq i \leq m \text{ and } 1 \leq j \leq N, \tag{3}
\]

where \( f \) is the mapping function approximated by the PointNet [38] neural network and \( g \) is a symmetric function representing the geometric feature of a point cloud \( \mathcal{X}_i \) and can be approximated as

\[
g(\mathcal{X}_i) = s(h(x_1, y_1), \ldots, h(x_N, y_N)) \forall (x_j, y_j) \in \mathcal{X}_i \text{ with } 1 \leq i \leq m \text{ and } 1 \leq j \leq N, \tag{4}
\]

where \( s \) is a symmetric function and \( h \) is a function representing the two shared Multilayer Perceptrons (MLPs) [30, 38, 39] in the first branch of PointNet [38] (see Fig. 2). In simple words, \( g(\mathcal{X}_i) \) can be thought as the global feature in the PIPN architecture shown in Fig. 2. We further explain the role of shared MLPs and the symmetric function in the following section.

We define \( n_s \) as a global scaling variable controlling the size of the PIPN. In the following, we denote batch size by \( B \), and reemphasize that “batch size” in this study is the number of domains fed into PIPN at each epoch.

In practice, the input of PIPN is a three-dimensional tensor (i.e., numeric array) of size \( B \times N \times 2 \). Afterward, there are two sequential shared MLPs with sizes of \( (n_s \times 64, n_s \times 64) \) and \( (n_s \times 64, n_s \times 128, n_s \times 1024) \), as displayed in Fig. 2. In the next step, the symmetric operator \( s \) forms the global feature with a size of \( n_s \times 1024 \). As exhibited in Fig. 2, the global feature is concatenated to the intermediate feature tensor of size \( B \times N \times (n_s \times 64) \), resulting in a tensor of size \( B \times N \times (n_s \times 1024) \). Next, two other sequential shared MLPs, respectively, with sizes of \( (n_s \times 512, n_s \times 256, n_s \times 128) \) and \( (n_s \times 128, n_{\text{pred}}) \) operate in the PIPN architecture. The output of the previous action is a tensor of size \( B \times N \times n_{\text{pred}} \), as shown in Fig. 2. Using automatic differentiation [1], the governing PDEs (Eqs. 1–2) are built and fed into the PIPN loss function. Note that reasonable choices for \( n_s \) could be 0.125, 0.25, 0.5, 1.0, 2.0, and 4.0 such that a positive integer number represents the size of resulting shared MLPs.

#### Shared MLPs and symmetric functions

The PIPN is invariant with respect to \( N! \) permutations of the input vector (see Fig. 2). In other words, if we randomly permute the input vector \( \mathcal{X}_i \), the constructed geometry of the domain does not change and thus the solution \( \mathcal{Y}_i \) should not change. PIPN becomes permutation invariant by means of two features: shared MLPs and symmetric functions.

The concept of shared MLPs should not be confused with regular fully connected layers or so-called “dense” layers in the TensorFlow [1] terminology. Here we explain the concept of shared MLPs with a simple example. Let us consider the first MLP layer in the first branch of PIPN with a size of \( (n_s \times 64, n_s \times 64) \) as shown in Fig. 2. Let us take \( n_s = 1 \) for a moment. More specifically, we focus on the first shared layer with a size of 64. The transpose of the input vector \( \mathcal{X}_i \) can be written as

\[
\mathcal{X}_i^{tr} = \begin{bmatrix} x_1 & x_2 & \ldots & x_N \\ y_1 & y_2 & \ldots & y_N \end{bmatrix}, \tag{5}
\]
After applying the first shared layer to $\mathcal{X}_i$, the output is a matrix of size $64 \times N$ and can be written as

$$
\begin{bmatrix}
a_{64 \times 1}^{(1)} & a_{64 \times 1}^{(2)} & \cdots & a_{64 \times 1}^{(N)}
\end{bmatrix},
$$

where $a_{64 \times 1}^{(1)}$, $a_{64 \times 1}^{(2)}$, ..., $a_{64 \times 1}^{(N)}$ are vectors, which are computed as follows:

$$
a_{64 \times 1}^{(1)} = \sigma \left( W_{64 \times 2} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + b_{64 \times 1} \right),
$$

$$
a_{64 \times 1}^{(2)} = \sigma \left( W_{64 \times 2} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} + b_{64 \times 1} \right),
$$

$$
\vdots
$$

$$
a_{64 \times 1}^{(N)} = \sigma \left( W_{64 \times 2} \begin{bmatrix} x_N \\ y_N \end{bmatrix} + b_{64 \times 1} \right),
$$

where $W$ and $b$ are the shared weight matrix and bias vector, respectively. The nonlinear activation function is shown by $\sigma$, which acts elementwise. As can be realized from Eq. 7, the same (shared) $W$ and $b$ are applied to each spatial point of the domain with the corresponding vector of $[x_i, y_i]^T$, while $1 \leq i \leq N$. That is why we call it shared MLPs. A similar procedure is conducted at the rest of the layers. By this strategy, it is immediately concluded that each point is independently processed in the PIPN framework and the only place that points meet each other is where the global feature needs to be identified (see Eq. 4).

Concerning the symmetric function, we consider the two formats: one being the “maximum” function such that

$$
g(\mathcal{X}_i) = \max(h(x_1, y_1), \ldots, h(x_N, y_N)); \forall (x_j, y_j) \in \mathcal{X}_i \text{ with } 1 \leq i \leq m \text{ and } 1 \leq j \leq N, \tag{8}
$$

and the other being the “average” function such that

$$
g(\mathcal{X}_i) = \text{average}(h(x_1, y_1), \ldots, h(x_N, y_N)); \forall (x_j, y_j) \in \mathcal{X}_i \text{ with } 1 \leq i \leq m \text{ and } 1 \leq j \leq N. \tag{9}
$$

We compare the PIPN performance using these two symmetric functions. One may refer to Ref. [25] for a further description of the details of PIPN; and to Ref. [38] for a further discussion on the computer science aspects of PointNet [38].

### 2.2.2 Loss function

For any pair of $\mathcal{X}_i$ and $\mathcal{Y}_i$ ($1 \leq i \leq m$), the residuals of the linear momentum in the $x$ direction ($J_i^{\text{momentum}_x}$), linear momentum in the $y$ direction ($J_i^{\text{momentum}_y}$), along with the residuals of the sparse observations of the displacement field ($J_i^{\text{displacement}_{\text{sensor}}}$), are respectively defined as follows:

$$
J_i^{\text{momentum}_x} = \frac{1}{N} \sum_{k=1}^{N} \left( \frac{\delta}{\delta x_k} \left( \frac{\delta \tilde{u}_k}{1 - \nu \delta x_k} + \frac{\nu \delta \tilde{v}_k}{1 - \nu \delta y_k} \right) - \frac{\delta}{\delta y_k} \left( \frac{1}{2} \frac{\delta \tilde{u}_k}{\delta y_k} + \frac{\delta \tilde{v}_k}{\delta x_k} \right) + \frac{\alpha}{1 - \nu \delta x_k} \frac{\partial T_k}{\partial x_k} \right)^2, \tag{10}
$$

$$
J_i^{\text{momentum}_y} = \frac{1}{N} \sum_{k=1}^{N} \left( \frac{\delta}{\delta y_k} \left( \nu \frac{\delta \tilde{u}_k}{1 - \nu \delta y_k} + \frac{\delta \tilde{v}_k}{1 - \nu \delta x_k} \right) - \frac{\delta}{\delta x_k} \left( \frac{1}{2} \frac{\delta \tilde{u}_k}{\delta y_k} + \frac{\delta \tilde{v}_k}{\delta x_k} \right) + \frac{\alpha}{1 - \nu \delta y_k} \frac{\partial T_k}{\partial y_k} \right)^2, \tag{11}
$$

$$
J_i^{\text{displacement}_{\text{sensor}}} = \frac{1}{M} \sum_{k=1}^{M} \left( \left( \tilde{u}_k - u_k^{\text{sensor}} \right)^2 + \left( \tilde{v}_k - v_k^{\text{sensor}} \right)^2 \right), \tag{12}
$$

5
where \( M \) is the number of sensors located at each point cloud for a sparse measurement of the displacement fields. The automatic differentiation operator is shown by \( \delta \). The \( x \) and \( y \) components of the displacement fields measured at the sensor locations are shown by \( u_{i}^{\text{sensor}} \) and \( v_{i}^{\text{sensor}} \), respectively, while the network outputs are denoted by \( \tilde{u}_{i} \) and \( \tilde{v}_{i} \). Note that because we assume that the temperature field is known to us, then the temperature gradient is also known. The hyperbolic activation function defined as

\[
\sigma(\gamma) = \frac{e^{\gamma} - 1}{e^{\gamma} + 1},
\]

is implemented in all the layers of PIPN, similar to Refs. [25, 26]. Note that due to the presence of the second-order spatial derivative of the displacement fields in the governing PDEs (Eqs. 1–2), the second-order derivative of the activation function used in PIPN must be well-defined.

As can be realized from Eq. 3, because the displacement fields are a function of \( g(X_{i}) \), the spatial derivatives of the displacement also become a function of \( g(X_{i}) \) in PIPN. For example,

\[
\frac{\delta u_{i}}{\delta x_{j}} = \frac{\delta f((x_{j}, y_{j}); g(X_{i}))}{\delta x_{j}}; \forall (x_{j}, y_{j}) \in X_{i} \text{ and } \forall u_{j} \in Y_{i} \text{ with } 1 \leq i \leq m \text{ and } 1 \leq j \leq N.
\]

Similar expressions can be written for \( \frac{\delta u_{i}}{\delta y_{j}}, \frac{\delta v_{i}}{\delta x_{j}}, \frac{\delta v_{i}}{\delta y_{j}} \), etc. Hence, all the components presented in the loss function of PIPN contain the geometric information of the point clouds of the set \( \Phi = \{V_{i}\}_{i=1}^{N} \), a specific feature that is not available in regular PINNs [42]. Eventually, the PIPN loss function is written as

\[
J = \frac{B}{m} \sum_{b=1}^{m/B} \left( \frac{1}{B} \sum_{i=1}^{hB/B}(1-\delta f((x_{j}, y_{j}); g(X_{i})) \delta_{X_{i}} \sum_{i=1}^{N} \mathcal{J}_{i}^{\text{momentum}} + \mathcal{J}_{i}^{\text{displacement}}) \right),
\]

where \( \omega_{\text{momentum}} \) and \( \omega_{\text{sensor}} \) are the corresponding weights of each residual, while their units are the inverse of the unit of their associated residuals such that the total loss function \( (J) \) becomes unitless at the end. We elaborate on the choice of these weight factors in the following section.

**Component weights in the PIPN loss function**

Based on our machine learning experiments, the choice of \( \omega_{\text{momentum}} \) and \( \omega_{\text{sensor}} \) remarkably affect the convergence rate of the training procedure. Here, we propose a few simple, but practical functions for \( \omega_{\text{momentum}} \) and \( \omega_{\text{sensor}} \), and discuss their effects in Sect. 3. The first trivial selection is setting an equal weight for both the momentum and sensor components of the loss function such that

\[
\left\{ \begin{array}{c}
\omega_{\text{momentum}} = 1 \text{ m} \\
\omega_{\text{sensor}} = 1 \text{ m}^{-1}.
\end{array} \right.
\]

The next choice is setting a higher weight for the sensor component compared to the momentum one in the PIPN loss function (see Eq. 13) such that

\[
\left\{ \begin{array}{c}
\omega_{\text{momentum}} = 1 \text{ m} \\
\omega_{\text{sensor}} = \omega_{0} \text{ m}^{-1}, \text{ with } \omega_{0} > 1.
\end{array} \right.
\]

Note that \( \omega_{0} \) remains constant during the training and can be thought of as a hyperparameter. Additionally, our machine learning experiments show that setting a higher weight for the momentum component compared to the sensor one leads to a significant reduction in the convergence speed and thus we do not propose it here. Our third proposal is setting a higher weight for the sensor component, while this weight exponentially decreases during the training process such that

\[
\left\{ \begin{array}{c}
\omega_{\text{momentum}} = 1 \text{ m} \\
\omega_{\text{sensor}} = \max(\omega_{1} \exp(-\text{epoch} \gamma_{1}), 1.0) \text{ m}^{-1}, \text{ with } \omega_{1} > 1 \text{ and } r_{1} > 0,
\end{array} \right.
\]

where “epoch” refers to the training iteration. Again, \( \omega_{1} \) and \( r_{1} \) are the hyperparameters. Our last suggestion is similar to our third one with the difference that the weight decreases logarithmically such that

\[
\left\{ \begin{array}{c}
\omega_{\text{momentum}} = 1 \text{ m} \\
\omega_{\text{sensor}} = \max(\omega_{2} \ln(-\text{epoch} + r_{2}), 1.0) \text{ m}^{-1}, \text{ with } \omega_{2} > 1 \text{ and } r_{2} > 0,
\end{array} \right.
\]

where \( \omega_{2} \) and \( r_{2} \) are the hyperparameters. Note that in all our proposals, \( \omega_{\text{sensor}} \) never becomes less than one. Moreover, the units of \( \omega_{\text{momentum}} \) and \( \omega_{\text{sensor}} \) are set such that \( J \) becomes unitless.

At the end of this subsection, we address the fact that setting optimal weights in the loss functions of deep learning models is an active research area by itself (see e.g., Ref. [53]); however, this is the first time that we introduce this concept to the PIPN configuration.
Table 1: Description of the generated geometries; $R$ indicates the radius of the circumscribed circle of the inner thin cavity. $\Omega$ denotes variation (in degree) in the orientation of the inner thin cavity with respect to its geometric center.

<table>
<thead>
<tr>
<th>The shape of the inner thin cavity</th>
<th>$R$</th>
<th>$\Omega$</th>
<th>Side length of thin square plates</th>
<th>Number of selected data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>0.35 m</td>
<td>1, 3, ..., 87, 89</td>
<td>1.6 m, 1.8 m, 2 m</td>
<td>134</td>
</tr>
<tr>
<td>Regular pentagon</td>
<td>0.30 m</td>
<td>1, 3, ..., 69, 71</td>
<td>1.6 m, 1.8 m, 2 m</td>
<td>107</td>
</tr>
<tr>
<td>Regular hexagon</td>
<td>0.30 m</td>
<td>1, 3, ..., 57, 59</td>
<td>1.6 m, 1.8 m, 2 m</td>
<td>90</td>
</tr>
<tr>
<td>Regular heptagon</td>
<td>0.30 m</td>
<td>1, 3, ..., 49, 51</td>
<td>1.6 m, 1.8 m, 2 m</td>
<td>76</td>
</tr>
<tr>
<td>Regular octagon</td>
<td>0.30 m</td>
<td>1, 3, ..., 43, 45</td>
<td>1.6 m, 1.8 m, 2 m</td>
<td>66</td>
</tr>
<tr>
<td>Regular nonagon</td>
<td>0.30 m</td>
<td>1, 3, ..., 37, 39</td>
<td>1.6 m, 1.8 m, 2 m</td>
<td>59</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>532</strong></td>
</tr>
</tbody>
</table>

Figure 3: Sensor locations for three domains from the set $\Phi = \{V_i\}_{i=1}^{532}$; Red triangles show sensors measuring the displacement fields.

2.3 Data generation

Computational domains of the set $\Phi = \{V_i\}_{i=1}^{m}$ are thin square plates with a cavity. The side length of the square plates takes three different values while the cavity takes shapes of a square, regular pentagon, regular hexagon, regular octagon, and regular nonagon. We further enlarge the number of domains by rotating the cavity with respect to the fixed thin square plates. Details of the geometric features of the set $\Phi = \{V_i\}_{i=1}^{m}$ are listed in Table 1. In total, 532 geometries (i.e., $m = 532$) are the input of PIPN. In this way, we establish the set $\Phi = \{V_i\}_{i=1}^{532}$. Three examples of geometries of this set are shown in Fig. 3. In our machine learning investigations, we consider batch sizes ($B$) of 7, 14, 19, 28, 38, 76, and 133, which are all divisors of 532. We recall that the maximum batch size ($B$) implemented by Kashefi and Mukerji [25] was 13 and the maximum number of input data ($m$) considered by Kashefi and Mukerji [25] was 108. Another difference is that the side length of the outer boundaries takes three values of 1.6 m, 1.8 m, and 2.0 m in the current study; however, Kashefi and Mukerji [25] exclusively considered a fixed value of 2.0 m for the side length of outer boundaries when they studied natural convection in a square enclosure with a cylinder (see Fig. 12 and Table 7 of Ref. [25]). In this way, we experience more variations in geometric features of the input point clouds in this study and thus a more challenging task is defined for PIPN.

The MATLAB PDE toolbox is used for the two purposes of validation of the PIPN predictions and generations of the sparse labeled data at virtual sensor locations. The ground truth data is generated using the toolbox as follows. The domains of the set $\Phi = \{V_i\}_{i=1}^{532}$ are loaded by thermal conduction with Dirichlet boundary conditions such that the temperature on the outer and inner boundaries are respectively set to 1 and zero in International Unit System. Note that the thermal conductivity does not affect the solution of the Laplace equation governing the steady-state thermal conduction. Moreover, we impose zero displacement boundary conditions on the boundaries of the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$.

2.4 Computational considerations

The Poisson ratio ($\nu$) is set to 0.3 and the thermal expansion coefficient ($\alpha$) is set to 1.0 in the International Unit System. Note that elastic Young’s modulus ($E$) does not appear in Eqs. (10–11) and thus is not a problem input. Furthermore, we set $N = 2021$ and $M = 81$ in the point clouds. The sensors are approximately equally spaced and located in the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$. The sensor locations for three different domains taken from the
Table 2: Error analysis of the displacement fields for the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$ when $B = 28$, $n_s = 1$, $\omega_{\text{sensor}} = 50$ m$^{-1}$, and $\omega_{\text{momentum}} = 1$ m. $||.||$ indicates the $L^2$ norm.

|       | $||u - u||/||u||$ | $||v - v||/||v||$ |
|-------|------------------|------------------|
| Average | 8.11108E−2       | 8.60237E−2       |
| Maximum | 5.74907E−2       | 6.32261E−2       |
| Minimum | 1.19782E−1       | 1.31120E−1       |

The set $\Phi = \{V_i\}_{i=1}^{532}$ are exhibited, for instance, in Fig. 3. We use the Adam optimizer [29] with the hyperparameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\dot{\epsilon} = 10^{-6}$. The mathematical explanation of $\beta_1$, $\beta_2$, and $\dot{\epsilon}$ are expressed in Ref. [29]. The constant learning rate of 0.0003 is set for the entire machine learning experiments. For a fair comparison, all the simulations are run on an NVIDIA A100 SXM4 graphic card with a memory clock rate of 1.41 GHz and 40 Gigabytes of RAM.

3 Results and discussion

3.1 General analysis

In Table 2, we tabulate the average, maximum, and minimum errors of the predicted displacement fields in the $x$ and $y$ directions for the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$ with batch size of $B = 28$, the network size of $n_s = 1.0$, and the weight of $\omega_{\text{sensor}} = 50$ m$^{-1}$ for the sensor component in the PIPN loss function (see Eq. 15). According to the data tabulated in Table 2, the average pointwise relative errors ($L^2$ norm) for all 532 geometries of the set $\Phi$ is less than 9%, indicating a successful accomplishment for the PIPN framework. Moreover, the maximum pointwise relative errors ($L^2$ norm) do not exceed 14%, demonstrating a reasonable accuracy for engineering applications.

A comparison between the ground truth (i.e., the finite element solutions) and the PIPN predictions is made in Figs. 4−6 for six different geometries taken from the set $\Phi = \{V_i\}_{i=1}^{532}$. As can be observed in Figs. 4−6, there is an excellent agreement between the ground truth and the displacement fields predicted by PIPN. For each geometry, the maximum local pointwise error happens on the boundaries of the inner cavity of thin plates. This outcome is expectable because, first, PIPN is not informed from boundary conditions (i.e., zero displacements), and second, most variations from one geometry to another one take place on the boundaries of the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$. In fact, predicting boundary values is the most difficult task for PIPN during the training procedure. This fact can be realized by looking at Fig. 7. We exhibit the prediction of PIPN for the displacement fields for one geometry taken from the set $\Phi = \{V_i\}_{i=1}^{532}$ after 3, 500, and 1500 epochs. Based on what can be seen in Fig. 7, the PIPN prediction is inaccurate after 10 epochs. After 500 epochs, the displacement fields are accurately predicted by PIPN except on the boundaries of the domain. An improvement in the prediction of zero displacements on the boundaries of the domain is observed at the PIPN outcome after 1500 epochs. Hence, we conclude that the most time-consuming part for PIPN is to predict the right values on the domain boundaries.

The absolute pointwise error ($L^2$ norm) for geometries with the maximum and minimum relative pointwise errors of the displacement fields in the $x$ and $y$ directions are shown in Fig. 8. According to Fig. 8, the extremum errors occur for domains with different geometries of the set $\Phi = \{V_i\}_{i=1}^{532}$, revealing the fact that PIPN is not overfitted to one specific geometry. Moreover, we observe that the maximum pointwise errors happen for the domains with the side length of 1.6 m, which is the smallest side length available in the data set $\Phi = \{V_i\}_{i=1}^{532}$. On the other hand, the minimum pointwise errors occur for geometries with the side square length of 2.0 m, which is the largest available side length in the data set $\Phi = \{V_i\}_{i=1}^{532}$. Based on our deep learning experiments, the ideal range for the spatial coordinates of the input geometries is when they are expanded in $[-1, 1]$. Having stretched geometries out of $[-1, 1]$ or compressed geometries inside of $[-1, 1]$ results in an error increase. Therefore, the domains with a side length of 1.6 m end up having higher levels of errors compared to larger side lengths.
Figure 4: The first group of examples taken from the set $\Phi = \{V_i\}_{i=1}^{532}$, comparing the finite element solutions to the PIPN predictions for the displacement fields.
Figure 5: The second group of examples taken from the set \( \Phi = \{ V_i \}_{i=1}^{532} \), comparing the finite element solutions to the PIPN predictions for the displacement fields.
Figure 6: The third group of examples taken from the set $\Phi = \{V_i\}_{i=1}^{532}$, comparing the finite element solutions to the PIPN predictions for the displacement fields.
According to Table 3, for a fixed network size of $n$, experiments are not doable such as the combination of a batch size of $B$ with $s_5$ = 0.25 where $\omega_{\text{sensor}} = 50$ m$^{-1}$ and $\omega_{\text{momentum}} = 1$ m. $||\cdot||$ denotes the $L^2$ norm. The cross symbol ($\times$) indicates that the machine learning experiment is not doable due to a memory limitation.

### Table 3: Error analysis of the displacement fields for the domains of the set $\Phi = \{V_i\}_{i=1}^{32}$ for different batch sizes ($B$) and three different network sizes of $n_s = 1.0$, $n_s = 0.5$, and $n_s = 0.25$ when $\omega_{\text{sensor}} = 50$ m$^{-1}$ and $\omega_{\text{momentum}} = 1$ m. $||\cdot||$ denotes the $L^2$ norm. The cross symbol ($\times$) indicates that the machine learning experiment is not doable due to a memory limitation.

| Batch size ($B$) | $n_s = 1.0$ Average $||\widehat{u} - u||/||u||$ | $n_s = 1.0$ Average $||\widehat{v} - v||/||v||$ | $n_s = 0.5$ Average $||\widehat{u} - u||/||u||$ | $n_s = 0.5$ Average $||\widehat{v} - v||/||v||$ | $n_s = 0.25$ Average $||\widehat{u} - u||/||u||$ | $n_s = 0.25$ Average $||\widehat{v} - v||/||v||$ |
|------------------|------------------------------------------|------------------------------------------|------------------------------------------|------------------------------------------|------------------------------------------|------------------------------------------|
| 7                | 8.56410E–2 $||\widehat{u} - u||/||u||$ | 8.58942E–2 $||\widehat{v} - v||/||v||$ | 8.40571E–2 $||\widehat{u} - u||/||u||$ | 8.90152E–2 $||\widehat{v} - v||/||v||$ | 1.00000 $||\widehat{u} - u||/||u||$ | 1.00000 $||\widehat{v} - v||/||v||$ |
| 14               | 8.24455E–2 $||\widehat{u} - u||/||u||$ | 7.84251E–2 $||\widehat{v} - v||/||v||$ | 8.71806E–2 $||\widehat{u} - u||/||u||$ | 7.68771E–2 $||\widehat{v} - v||/||v||$ | 9.99999E–1 $||\widehat{u} - u||/||u||$ | 1.00000 $||\widehat{v} - v||/||v||$ |
| 19               | 8.34842E–2 $||\widehat{u} - u||/||u||$ | 8.42466E–2 $||\widehat{v} - v||/||v||$ | 8.03585E–2 $||\widehat{u} - u||/||u||$ | 8.33552E–2 $||\widehat{v} - v||/||v||$ | 5.08574E–1 $||\widehat{u} - u||/||u||$ | 1.00000 $||\widehat{v} - v||/||v||$ |
| 28               | 8.11108E–2 $||\widehat{u} - u||/||u||$ | 8.60237E–2 $||\widehat{v} - v||/||v||$ | 7.44200E–2 $||\widehat{u} - u||/||u||$ | 7.40376E–2 $||\widehat{v} - v||/||v||$ | 3.24522E–1 $||\widehat{u} - u||/||u||$ | 3.45531E–1 $||\widehat{v} - v||/||v||$ |
| 38               | 8.69939E–2 $||\widehat{u} - u||/||u||$ | 9.05297E–2 $||\widehat{v} - v||/||v||$ | 8.45176E–2 $||\widehat{u} - u||/||u||$ | 8.20743E–2 $||\widehat{v} - v||/||v||$ | 3.68147E–1 $||\widehat{u} - u||/||u||$ | 3.69285E–1 $||\widehat{v} - v||/||v||$ |
| 76               | $\times$ $||\widehat{u} - u||/||u||$ | $\times$ $||\widehat{v} - v||/||v||$ | $\times$ $||\widehat{u} - u||/||u||$ | $\times$ $||\widehat{v} - v||/||v||$ | $\times$ $||\widehat{u} - u||/||u||$ | $\times$ $||\widehat{v} - v||/||v||$ |
| 133              | $\times$ $||\widehat{u} - u||/||u||$ | $\times$ $||\widehat{v} - v||/||v||$ | $\times$ $||\widehat{u} - u||/||u||$ | $\times$ $||\widehat{v} - v||/||v||$ | $\times$ $||\widehat{u} - u||/||u||$ | $\times$ $||\widehat{v} - v||/||v||$ |

### 3.2 Effect of network size and batch size

We first explore the effect of batch sizes ($B$) on the PIPN performance in this subsection. It is worthwhile to note that at each epoch, the geometries are shuffled and randomly divided into mini-batch, exactly similar to the scenario of fully supervised learning models. We collect the average relative pointwise errors ($L^2$ norm) of the displacement fields for the batch sizes ($B$) of 7, 14, 19, 28, 38, 76, and 133 for three different network sizes ($n_s$) of 1.0, 0.5, and 0.25 in Table 3. Note that due to the memory limitations, a few of these machine learning experiments are not doable such as the combination of a batch size of $B = 133$ with the network size of $n_s = 1.0$. According to Table 3, for a fixed network size of $n_s = 1.0$, the batch size ($B$) does not significantly affect the prediction accuracy. A similar story is true when the network size of $n_s = 0.5$ is taken, except for the batch size of $B = 76$, where the accuracy of the PIPN prediction notably decreases and the average relative pointwise errors ($L^2$ norm) of the displacement fields becomes approximately 37%. The optimal batch size ($B$) is 28 for this network size ($n_s = 0.5$). By reducing the network size to $n_s = 0.25$, the overall performance of PIPN sharply decreases. For batch sizes ($B$) of 7, 14, 19, 76, and 133, the PIPN prediction is completely off. For the batch sizes ($B$) of 28 and 38, although the average relative pointwise errors ($L^2$ norm) are less than 50%, the PIPN solution is not reliable. The lack of PIPN performance for the choice of $n_s = 0.25$ is further discussed at the end of this subsection. But it is concluded that it is important to first select a PIPN with a suitable size, before investigating the effect of the batch size ($B$). In other words, the effect of network size on PIPN is significantly more noticeable compared to the influence of the batch size.
Figure 8: Distribution of absolute pointwise error when the relative pointwise error ($L^2$ norm) becomes (a) maximum for $\tilde{u}$, (b) maximum for $\tilde{v}$, (c) minimum for $\tilde{u}$, and (d) minimum for $\tilde{v}$.

Table 4: Error analysis of the displacement fields for the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$ for different network sizes $(n_s)$ when $B = 19$, $\omega_{\text{sensor}} = 50 \text{ m}^{-1}$, and $\omega_{\text{momentum}} = 1 \text{ m}$. $||\ldots||$ indicates the $L^2$ norm.

| Network size $(n_s)$ | Average $||\tilde{u} - u||$ | Average $||\tilde{v} - v||$ |
|---------------------|-----------------------------|-----------------------------|
| 0.125               | 1.00001                     | 1.00000                     |
| 0.25                | 5.08574E-1                  | 1.00000                     |
| 0.5                 | 8.03585E-2                  | 8.33552E-2                  |
| 1.0                 | 8.34842E-2                  | 8.42466E-2                  |
| 1.5                 | 8.51296E-2                  | 7.35883E-2                  |
| 2.0                 | 9.25092E-2                  | 1.00152E-1                  |
| 2.5                 | 1.51389E-1                  | 1.52311E-1                  |

The evolution of the PIPN loss function (see Eq. 15) for the batch sizes ($B$) of 7, 14, 19, 28, 38, and 76 for the network size of $n_s = 0.5$ is shown in Fig. 9a. According to Fig. 9a, for the batch sizes ($B$) of 7, 14, 19, 28, and 38, although the loss function eventually converges to approximately the same value, the machine learning experiment with the batch size of $B = 19$ reaches this convergence with a significantly smaller number of epochs. Moreover, the PIPN loss associated with the batch size of $B = 76$ converges to a value larger than the other investigated batch sizes, as can be realized from Fig. 9a. This fact can be seen in another form by looking at the relative errors tabulated in Table 3 as already discussed in Table 3.

Additionally, the effect of network size is carried out and outcomes are listed in Table 4. Based on the information listed in Table 4, for relatively small network sizes ($n_s = 0.125, n_s = 0.5$) and relatively large network sizes ($n_s = 2.0, n_s = 2.5$), the PIPN configuration experiences a higher level of errors compared to the moderate sizes of the network ($n_s = 0.5, n_s = 1.0, n_s = 1.5$). For a relatively small PIPN, the network is too simple and a bias (i.e., underfitting) takes place. For a relatively large PIPN, the number of data is not sufficient to well determine the weight matrix ($W$) and bias vector ($b$) of the network, and hence, the PIPN predictions suffer from the lack of accuracy.

Furthermore, the evolution of the total loss value over all the geometries of the set $\Phi = \{V_i\}_{i=1}^{532}$ is displayed for the network sizes ($n_s$) of 0.125, 0.25, 0.5, 1.0, 1.5, 2.0, and 2.5 in Fig. 9b. As can be seen in Fig. 9b, for
the network size \((n_s)\) of 0.125, the training loss does not converge, leading to 100% relative errors as tabulated in Table 4. As discussed earlier, for the network sizes \((n_s)\) of 0.5, 1.0, and 1.5, the relative errors of PIPN are approximately the same and smaller than other network size choices. Among these three selections, the network size \((n_s)\) of 1.5 converges approximately after 800 epochs, while PIPN with the network size \((n_s)\) of 0.5 converges after approximately 2800 epochs. For the network size \((n_s)\) of 2.0 and 2.5, although the training loss converges, we observe a tremendous tendency for divergence and instability in the optimization process. The reason comes back to the fact that the number of data is not adequate for this choice of the PIPN size as explained in the previous paragraph.

### 3.3 Choice of a symmetric function in PIPN

We investigate the effectiveness of two different symmetric functions (see Eqs. 8–9) in the setting of the PIPN configuration. Specifically, we set the batch size of \(B = 19\) and the network size of \(n_s = 1.0\) with the weight of \(\omega_{\text{sensor}} = 50 \, \text{m}^{-1}\) for the sensor component in the PIPN loss function for this machine learning experiment. Figure 9c illustrates the evolution of the PIPN loss function (see Eq. 15) for the maximum and average as two symmetric functions for extracting global features of point clouds. For a fair comparison, the network size and the batch size are set the same in both experiments. As can be observed in Fig. 9c, the max function shows a better final performance compared to the average functions, though the loss with the average function initially drops faster than the loss function with the max function. However, a sharp decrease in the PIPN loss value with the maximum function occurs around approximately 1800 epochs such that this loss value becomes smaller than
Figure 10: Evolution of the weight of the sensor component ($\omega_{\text{sensor}}$) in the PIPN loss function during the training procedure for four different formulas (see Eqs. 16–19)

We note that Qi, et al. [38] and Qi, et al. [39] used the maximum function for the purpose of object classification and segmentations in the area of computer graphics, and we observe here a higher performance with the maximum function in the area of computational mechanics as well. The reason can be explained as follows. As shown by Kashefi, et al. [27] for supervised learning, using the maximum function, boundary points of point clouds of the domains always contribute to the global feature in PointNet [38]. This is while the most variation from one geometry to another one happens also on boundaries. In this way, PIPN can more clearly distinguish the PDE solution depending on the geometric features of each domain of the set $\Phi = \{V_i\}_{i=1}^{532}$.

### 3.4 Effect of component weights in the PIPN loss function

We investigate four different formulas for the weight component of the sensor data in the PIPN loss function. More specifically, we set $\omega_0 = 50$ m$^{-1}$ in Eq. 17, we set $\omega_1 = 50$ m$^{-1}$ and $r_1 = 800$ in Eq. 18, and we set $\omega_2 = \frac{50}{8}$ m$^{-1}$ and $r_2 = 3002$ in Eq. 19. We display the evolution of the weight of the sensor component ($\omega_{\text{sensor}}$) during training in Fig. 10 for these four different setups (Eqs. 16–19). The relative pointwise errors ($L^2$ norm) are tabulated in Table 5. In general, the best PIPN performances are obtained when the sensor component ($\omega_{\text{sensor}}$) has a constant but higher value compared to the PDE component ($\omega_{\text{momentum}}$) (see Eq. 17 and Fig. 12) and when the sensor component ($\omega_{\text{sensor}}$) logarithmically decreases while still has a significantly higher value compared to the PDE component ($\omega_{\text{momentum}}$) during most of the training procedure (see Eq. 18 and Fig. 12). Additionally, we plot the evolution of the PIPN loss function during training as a result of these four different formulas (see Eqs. 16–19). These results demonstrate that setting a reasonably higher weight for the sensor component of the loss function (i.e., $\omega_{\text{sensor}}$) compared to the PDE component of the loss function (i.e., $\omega_{\text{momentum}}$) even during the entire training procedure leads to more accurate outputs. Mathematically, because PIPN is not informed of any boundary conditions in this problem, there are infinite solutions to the governing PDEs and thus obtaining the right solution is challenging for PIPN. But by setting a higher weight for the sensor data, PIPN is forced to have a priority for satisfying the mismatch between the network output and sensor data; however, this enforcement makes it convenient to also satisfy the governing PDEs because PIPN looks for the PDE solutions in the space created by the sensor data (rather than a weakly unknown space).
Table 5: Error analysis of the displacement fields for the domains of the set $\Phi = \{V_i\}_{i=1}^{532}$ for different settings of weights $\omega_{\text{sensor}}$ in the loss PIPN function when $B = 28$ and $\omega_{\text{momentum}} = 1$. $||\ldots||$ shows the $L^2$ norm.

| $\omega_{\text{sensor}}$ | Average $||u^i - u||$ | Average $||v^i||$ |
|--------------------------|------------------------|-------------------|
| 1.0 m$^{-1}$             | 3.72051E−1             | 3.75747E−1        |
| 50.0 m$^{-1}$            | 8.11108E−2             | 8.60237E−2        |
| $\max\left(50 \times \exp\left(\frac{-\text{epoch}}{800}\right), 1.0\right)$ m$^{-1}$ | 3.71722E−1 | 3.71195E−1 |
| $\max\left(\frac{50}{\pi} \times \ln\left(\text{epoch} + 3002\right), 1.0\right)$ m$^{-1}$ | 9.81245E−2 | 9.63949E−2 |

Additionally, we plot the evolution of the total PIPN loss over all geometries of the set $\Phi = \{V_i\}_{i=1}^{532}$ for four different setups of the sensor component weight (see Eqs. 16–19) in Fig. 9d. Comparing the deep learning experiment with the constant value of $\omega_{\text{sensor}} = 50$ m$^{-1}$ (see Eq. 17) and the logarithmic function (see Eq. 19), a convergence with fewer epochs happens for the logarithmic function, as can be seen in Fig. 9d. Note that according to Table 5, PIPN demonstrates a higher performance for the two mentioned choices of $\omega_{\text{sensor}}$ in comparison with the constant value of $\omega_{\text{sensor}} = 1$ m$^{-1}$ (see Eq. 16) and the exponential function (see Eq. 18). Moreover, the PIPN loss evolution with the exponential function for presenting $\omega_{\text{sensor}}$ (see Eq. 18) has a smooth decay and converges to a value, which is slightly higher than the convergence values of deep learning experiments with the constant value of $\omega_{\text{sensor}} = 50$ m$^{-1}$ (see Eq. 17) and the logarithmic function (see Eq. 19). For the choice of the constant value of $\omega_{\text{sensor}} = 1$ m$^{-1}$ (see Eq. 16), the initial loss is smaller than the other three options simply because the weight of the sensor component in the loss function (see Eq. 15) is scaled down by a factor of 50. However, the initial loss decreases with a very gentle slope up to 3000 epochs and thus there is a 38% average relative error as reported in Table 5.

4 Summary and conclusions

Generally speaking, there are two different deep learning categories in the area of computational mechanics: weakly supervised learning models (requiring sparse labeled data) (see e.g., [42]) and fully supervised learning models (requiring plentiful labeled data) [48]. A specific class of supervised learning models is physics-informed neural networks (PINNs) [42]. From an industrial point of view, an ideal deep learning framework should contain the ability for predicting desired fields over hundreds (or even thousands) of domains with various geometries for the goal of swiftly optimizing geometric designs. Regular PINNs are applicable to the prediction of desired fields on a single geometry, whereas fully supervised learning models are used to predict desired fields on a few hundred geometries. In this sense, there is a gap between the weakly supervised learning and supervised learning models. Physics-informed PointNet (PIPN) [25] is a novel class of physics-informed deep learning algorithms that fills this gap. PIPN requires sparse label data but potentially is able to predict desired fields on a few hundred geometries.

In 2022, Kashefi and Mukerji [25] proposed PIPN and employed it to solve incompressible flow and thermal fields over 108 domains with different geometries, simultaneously. Furthermore, they only investigated the batch sizes of 1, 2, 3, and 4. In the current article, we tried to explore the underlying capacity of PIPN in terms of the number of geometries that desired fields can be predicted on, simultaneously, and also to investigate if PIPN (as a weakly supervised learning model) is able to compete with fully supervised learning models from this point of view or not. We answered this question by considering a linear elasticity problem and more specifically plane stress conditions.

Given our computational resources, PIPN was able to successfully predict the displacement fields over 532 domains with different geometries, simultaneously. The average relative pointwise error ($L^2$ norm) was approximately 9% over the set. For the first time, we comprehensively explored the effects of batch size on the PIPN performance. By the term “batch size”, we meant the number of geometries that were fed into PIPN at each sub-epoch. Particularly, we executed batch sizes of 7, 14, 19, 28, 38, 76, and 133. In addition, we pioneered introducing a global parameter for controlling the network size of PIPN. Moreover, we investigated the effect of this parameter on the accuracy of the PIPN predictions. It was concluded that the network size plays a more important role in controlling the PIPN performance compared to the batch size. It was observed that the network size should be compatible with the size of data (i.e., the number of geometries). In fact, we realized that when a suitable size for PIPN was selected, the effect of batch size on the output predictions by PIPN was insignificant. However, the batch size affected the convergence rate. Furthermore, we studied the accuracy of the displacement fields predicted by the PIPN methodology as a result of different constant weights and dynamics weights (i.e., as a function of epoch) for the component of the partial differential equations and the component of the sparse labeled data in the loss function. It was concluded that setting a constant higher weight for the component of the sparse labeled data compared to the component of the partial differential equations leads to higher prediction accuracy.
Declaration of competing interest

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

Data availability

The data and the developed Python code are available on the following GitHub repository: https://github.com/Ali-Stanford/PhysicsInformedPointNetElasticity

Acknowledgements

The authors acknowledge funding by Shell-Stanford Collaborative Project on Digital Rock Physics 2.0 for supporting this research project. Additionally, we would like to thank the Stanford Research Computing Center for supporting our studies by providing computing resources.

References


