PHYSICS-INFORMED NEURAL NETWORKS: A DEEP LEARNING FRAMEWORK FOR SOLVING INVERSE PROBLEMS AND APPLICATIONS IN MEDICAL IMAGING*

CHANGLIANG WEI , ELENI MICHAELIDOU , ALANNAH HOUNAT , AND AYSU ISMAYILOVA

Abstract. In recent years, deep neural networks (DNNs) have been predominant in solving inverse problems, i.e., recovering parameters from a parameterized model from observed data. One such approach, Physics Informed Neural Networks (PINNs), is particularly well adapted for solving PDE-based inverse problems. In this report, a general description of PINNS is provided. Specifically, using automatic differentiation, PINNs incorporate a differential equation into the neural network loss function. This report aims to explore the capacity of PINNs to retrieve the parameters of a given differential equation from simulated boundary data. The PINN algorithm can easily be applied to different types of differential equations. Specifically, the effectiveness of PINNs in solving first- and second-order linear and non-linear ordinary differential equations (ODEs) is demonstrated. In addition, PINNs can successfully be applied in higher dimensions. In particular, the application of PINNs to the two-dimensional Laplace equation is presented. Lastly, the potential use of the proposed framework to solve inverse problems in medical imaging, such as electric impedance tomography, is demonstrated.

1. Introduction. Inverse problems are a class of problems where the goal is to determine the unknown parameters or states of a system based on observed data. In many scientific and engineering applications, inverse problems play a crucial role in understanding and predicting the behavior of complex systems. However, solving inverse problems can be challenging due to the ill-posed nature of the problem, where small errors in the data or measurements can lead to large uncertainties in the estimated parameters [15].

Recently, PINNs have emerged as a promising approach for solving inverse problems. PINNs combine the power of neural networks with the physical knowledge of the problem to accurately estimate the unknown parameters or states, even in the presence of noisy and sparse data [15]. PINNs have been successfully applied in a wide range of fields, including medical imaging, geophysics, materials science, and fluid dynamics.

The rest of this report is structured as follows. Section 2 discusses the problem

*RSCAM Group Project, Supervisor: Matias Ruiz

statement for inverse problems for partial differential equations (PDEs) using PINNs. The details of the proposed approach adapted for solving PDE-based inverse problems are presented in Section 3. In addition, a brief introduction to the architecture of a deep neural network is made, and the formulation of the loss function and the PINN algorithm for solving differential equations is presented. Section 4 presents the computational results of using PINNs for solving ODEs. Specifically, the performance of PINNs in solving first-order and second-order linear and nonlinear ODEs is discussed, along with the application of PINNs to well-known PDEs such as the wave equation. Lastly, Section 5 highlights the use of PINNs in the specific application of electrical impedance tomography, a medical imaging technique that aims to reconstruct the internal conductivity distribution of a body from boundary measurements. The results of using PINNs for electrical impedance tomography are discussed.

2. Problem setup. This report examines the inverse problem for PDEs (data-driven discovery of PDEs) using PINNs. Specifically, the capacity of PINNs to retrieve the parameters of a given differential equation from simulated boundary data is explored. It should be noted that the statement of the inverse problem is as follows: given the solution of PDE (with unknown parameters λ) at a few sampling points, what is the value of model parameters λ that best describe the observed data [4]. This type of problem is data-driven; therefore, prior knowledge and information of the data are required.

3. Methodology - Physics-informed neural networks. This section briefly describes the anatomy of DNNs. In addition, the PINN framework adapted for solving inverse problems for differential equations is presented.

Notation. In the following, vectors of constants $\mathbf{x}$ are denoted in lowercase boldface and matrices $\mathbf{X}$ in capital boldface.

3.1. Deep neural network architecture. A DNN is a collection of neurons organized in multiple layers, where each neuron in a layer takes the neuron activations from the previous layers as input, performs a computation, and passes the output to the neurons on the next layer[12]. It should be noted that an L-layer neural network or a neural network of depth L corresponds to a network with an input layer (0th layer), $L-1$ hidden layers and an output layer (Lth layer) [8, 12]. Therefore, the number of layers refers to the layers of weights (hidden and output units)[2].

Let $\Lambda^L(\mathbf{x};\boldsymbol{\theta}) : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ be a NN of length L parameterized by θ, where $\boldsymbol{\theta} = \{\mathbf{W}^\ell, \mathbf{b}^\ell\}_{\ell=1}^L$ are the training parameters in the network, including both the weights and biases of the neural network. In addition, let N_ℓ be the number of neurons in the ℓth layer ($N_0 = d_{\text{in}}$, and $N_L = d_{\text{in}}$). Therefore, the set of weights and biases associated with the ℓth layer are denoted by $\mathbf{W}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$, respectively[8]. The neurons of the neural network jointly implement a complex nonlinear mapping from the input to the output [12]. Specifically, the nonlinear activation function σ is applied elementwise at layer ℓ before sending it as an input to the next layer $\ell + 1$. It is mentioned that common choices for the activation function σ are: a) the hyperbolic tangent tanh, b) the sigmoid function, and c) ReLU.

It should be noted that the simplest type of neural network is the feedforward neural network (FNN), where linear and nonlinear transformations are applied to the inputs recursively; therefore, the information is only passed forward[11]. This report employs FNN, which is suitable for most PDE problems. Hence, the FNN algorithm for computing the output Λ^L is defined as [8]:

$$\begin{aligned}
\text{input layer:} \quad & \Lambda^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^{d_{\text{in}}}, \\
\text{hidden layers:} \quad & \Lambda^\ell(\mathbf{x}) = \sigma(\mathbf{W}^\ell \Lambda^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell) \in \mathbb{R}^{N_\ell}, \quad \text{for } l = \{1, \ldots, L-1\}, \\
\text{output layer:} \quad & \Lambda^L(\mathbf{x}) = \mathbf{W}^L \Lambda^{L-1}(\mathbf{x}) + \mathbf{b}^L \in \mathbb{R}^{d_{\text{out}}}.
\end{aligned}$$

As mentioned, the collection of weights and biases (parameter vector $\boldsymbol{\theta}$) are the learnt parameters of the neural network. This is achieved by minimizing a loss function, $\mathcal{L}$. Common choices for the loss function are the residual sum-of squares

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{m} (\Lambda^L(\mathbf{x_i};\boldsymbol{\theta}) - u_i)^2$$

and the mean squared error loss

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\Lambda^L(\mathbf{x_i};\boldsymbol{\theta}), u_i),$$

where $\mathbf{x} \in \mathbb{R}^m$ is the training set, and u_i and $\Lambda^L(\mathbf{x_i};\boldsymbol{\theta})$ are the target value and the corresponding predicted value, respectively [6].

3.2. Physics-informed neural networks (PINNs). PINNs can accurately solve forward and inverse problems where the solution of the differential equation and the parameters involved in the differential equation, respectively, are obtained from

training data, i.e., $\mathcal{T} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_{|\mathcal{T}|}\}$ of size $|\mathcal{T}|$ [11]. In the PINN algorithm for solving differential equations, the neural network is restricted to satisfy the physics imposed by the differential equation and boundary/initial conditions. Therefore, PINNs enforce information about a given differential equation by adding equations constraint in the neural network loss function. Specifically, the loss function is computed using the contribution from the neural network part corresponding to the boundary/initial conditions (see Sec 3.1) and the residual from the differential equation (ODE or PDE) provided by the physics-informed part [8].

The objective is to construct a neural network, $\Lambda^L(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$, to satisfy both the physics imposed by the differential equation, and the boundary/initial conditions. Equivalently, the aim is to find the optimal parameters $\boldsymbol{\theta}$ of the model that minimize the appropriately defined loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})$.

Consider the following differential equation parameterized by $\boldsymbol{\lambda}$:

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega, \tag{3.1}$$

with appropriate boundary conditions[1]

$$\mathcal{B}(u, \mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega,$$

where $u(\mathbf{x})$ is the solution of the system to be computed with $\mathbf{x} = (x_1, \ldots, x_d)$ an input vector defined on a domain $\Omega \subset \mathbb{R}^d$ [11]. The solution of the differential equation (3.1) is approximated by a DNN. Specifically, a neural network, $\Lambda^L(\mathbf{x}; \boldsymbol{\theta})$, is constructed to approximate the solution of the physical system, i.e., $\Lambda^L(\mathbf{x}; \boldsymbol{\theta}) \approx u(\mathbf{x})$. The neural network takes the input $\mathbf{x}$ and outputs a vector with the same dimensions as u [11]. By choosing the neural network to be the approximation of the differential equation, the derivatives of $\Lambda^L(\mathbf{x}; \boldsymbol{\theta})$ can be computed with respect to its input $\mathbf{x}$ by applying the chain rule for differentiating compositions of functions using automatic differentiation (AD) [11]. It should be noted that machine learning libraries such as PyTorch provide sufficient implementations of AD techniques [14].

As mentioned, the neural network is restricted to satisfy the physics imposed by the differential equation and boundary conditions. Let $\mathcal{T} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_{|\mathcal{T}|}\}$ of size $|\mathcal{T}|$ be the training data, where $\mathcal{T}_u = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_{|\mathcal{T}_\mathbf{u}|}\} \subset \partial\Omega$ and $\mathcal{T}_f = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_{|\mathcal{T}_\mathbf{f}|}\} \subset \Omega$ are points on the boundary and in the domain, respectively

[1] The initial condition can be treated as a Dirichlet boundary condition on the spatial domain.

[11]. Hence, the neural network $\Lambda^L(\mathbf{x};\boldsymbol{\theta})$ is restricted on those points (set of residual points). Therefore the parameters $\boldsymbol{\theta}$ of the model can be learned by minimizing the loss function defined as sum of mean square error loss of the differential equation and the boundary condition residuals:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = \mathcal{L}_u(\boldsymbol{\theta};\mathcal{T}_u) + \mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f), \tag{3.2}$$

where

$$\mathcal{L}_u(\boldsymbol{\theta};\mathcal{T}_u) = \frac{1}{|\mathcal{T}_u|}\sum_{\mathbf{x}\in\mathcal{T}_u}\left\|u(\mathbf{x}) - \Lambda^L(\mathbf{x};\boldsymbol{\theta})\right\|^2,$$

$$\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|}\sum_{\mathbf{x}\in\mathcal{T}_f}\left\| f\left(\mathbf{x}; \frac{\partial\Lambda^L}{\partial x_1},\ldots,\frac{\partial\Lambda^L}{\partial x_d}; \frac{\partial^2\Lambda^L}{\partial x_1\partial x_1},\ldots,\frac{\partial^2\Lambda^L}{\partial x_1\partial x_d};\ldots;\boldsymbol{\lambda}\right)\right\|^2.$$

The loss $\mathcal{L}_u(\boldsymbol{\theta};\mathcal{T}_u)$ corresponds to the boundary data while $\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f)$ enforces the structure (physics) imposed by the differential equation (3.1) at the set of points $\mathcal{T}_f$ [15]. Hence, the best parameters $\boldsymbol{\theta}^*$ that minimizes $\mathcal{L}(\boldsymbol{\theta};\mathcal{T})$ are expressed as $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta};\mathcal{T})$.

The algorithm of PINN is presented in Algorithm 3.1 [11].

Algorithm 3.1 The PINN algorithm for solving differential equations

Input: training data $\mathcal{T}$.

Output: training parameters of the model $\boldsymbol{\theta}$.

1: Build a neural network $\Lambda^L(\mathbf{x};\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$.
2: Specify the two training sets $\mathcal{T}_f$ and $\mathcal{T}_b$ for the equation and boundary conditions.
3: Specify a loss function by summing the mean square error loss of the differential equation and the boundary condition residuals.
4: Train the neural network to find the best parameters $\boldsymbol{\theta}^*$ by minimizing the loss function $\mathcal{L}(\boldsymbol{\theta};\mathcal{T})$.

3.3. PINNs for solving inverse problems. As mentioned, the main focus of the report is to explore the capacity of PINNs to retrieve the parameters of a given differential equation from simulated boundary data. Therefore, the PINN algorithm can easily be adapted for solving PDE-based inverse problems. Specifically, in an inverse problem, the parameters $\boldsymbol{\lambda}$ in (3.1) are considered unknown. Hence, in addition to information on the boundary conditions ($\mathcal{T}_u$) and the differential equation ($\mathcal{T}_f$), extra information on some data points $\mathcal{T}_i \subset \Omega$ is provided; $\mathcal{I}(u,\mathbf{x}) = 0$ for $\mathbf{x} \in \mathcal{T}_i$ [11].

Therefore, in an inverse problem, the collection of parameters of the model $\boldsymbol{\theta}$ along with the parameters $\boldsymbol{\lambda}$ of the differential operator, can be learned by minimizing the loss function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}) = \mathcal{L}_u(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_u) + \mathcal{L}_f(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_f) + \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i), \tag{3.3}$$

where

$$\mathcal{L}_i(\boldsymbol{\theta}; \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{x} \in \mathcal{T}_i} \left\| \mathcal{I}(\Lambda^L, \mathbf{x}) \right\|^2.$$

It should be noted that the only difference between the loss function for solving forward problems (3.2) and the loss function for solving inverse problems (3.3) is the additional term $\mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i)$ [11]. Hence, the best parameters $\boldsymbol{\theta}^*$ and $\boldsymbol{\lambda}^*$ are obtained as $\boldsymbol{\theta}^*, \boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\theta}, \boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T})$.

4. Computational results. This section presents the outcomes of applying PINNs to solve first-order and second-order linear and non-linear ODEs and the two-dimensional Laplace equation. It includes a detailed analysis of the accuracy of the PINNs framework. Specifically, the overall performance of the proposed neural network architecture is examined in each type of differential equation. Moreover, the effectiveness of the model in identifying the unknown parameters of the differential equation when the training data is captured with and without noise is demonstrated. Specifically, the noise data are generated by 1% of Gaussian uncorrelated noise corruption.

A fully connected network architecture is used for the numerical experiments with 20 neurons per hidden layer and the hyperbolic tangent (tanh) activation function before all hidden and output layers. Lastly, the L-BFGS optimizer is used to minimize the loss function. It should be noted that the network architecture in each example is different. Specifically, a different number of hidden layers is examined for model selection. The final chosen model is the neural network which accurately predicts the model parameters $\boldsymbol{\lambda}$ having fewer hidden layers. Lastly, it should be noted that 50 training data and the appropriate number of boundary data are provided as input to the neural network in each numerical example.

4.1. Ordinary differential equations. Different ODEs are examined to evaluate the performance of PINNs in identifying the unknown parameters of the equation.

4.1.1. First-order linear and nonlinear ODEs. Let's consider the following first-order linear ODE:

$$\frac{du}{dx} + (x + \frac{1+3x^2}{1+x+x^3})u = x^3 + 2x + x^2\frac{1+3x^2}{1+x+x^3}$$

with boundary condition $u(0) = 1$, and $x \in [0, 1]$.

The differential equation $\mathcal{D}(x; \lambda)$ of PINNs is defined by

$$\mathcal{D}(x; \lambda) \coloneqq \frac{du}{dx} + (\lambda_1 x + \frac{1+\lambda_2 x^2}{1+x+x^3})u = x^3 + 2x + x^2\frac{1+\lambda_2 x^2}{1+x+x^3},$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$. A neural network $\Lambda(x; \theta)$ parameterized by θ approximates the solution $u(x)$. When $\lambda_1 = 1$ and $\lambda_2 = 3$ the exact solution is defined as $u(x) = e^{(-x^2/2)/(1+x+x^3)} + x^2$.

A neural network $\Lambda(x; \theta)$ parameterized by θ is defined to approximate the solution $u(x)$ of the first-order linear ODE. The loss function of the neural network $\Lambda(x; \theta)$ can be defined as:

$$\mathcal{L}(\theta, \lambda_1; \mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} \|u_i - \Lambda(x_i; \theta)\|^2 + \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{D}(x_i; \lambda_1)\|^2,$$

where $\mathcal{T} = (x_1, u_1), (x_2, u_2), \ldots, (x_N, u_N)$ is the training set, with N pairs of points in the domain of interest.

In this example, 100 equally spaced points in the range $(0, 1)$ are generated for variables $\mathbf{x} = [x_1, x_2, \ldots, x_{100}]$ and $\mathbf{u} = [u_1, u_2, \ldots, u_{100}]$, where u_i is generated as $u_i = e^{(-x_i^2/2)/(1+x_i+x_i^3)} + x_i^2$ for $i = 1, 2, \ldots, 100$. A training set $\mathcal{T}_f$ of 50 pairs of points is randomly selected from $\mathbf{x}$ and $\mathbf{u}$ to enforce the differential equation $\mathcal{D}(x; \lambda)$, where $\mathcal{T}_f = (x_1, u_1), (x_2, u_2), \ldots, (x_{50}, u_{50})$. The combined training set $\mathcal{T} = \mathcal{T}_b \cup \mathcal{T}_f$ is then used to train five different classes of neural networks, each with a different number of hidden layers (1, 2, 4, 7, and 10 hidden layers) with 20 neurons in each hidden layer.

The results are shown in Figure B.1. The first two plots show that the error of the parameter (λ_1, λ_2) becomes close to 0 through the iterations, indicating that the parameters are getting closer to the exact values. All five classes of neural networks perform well (predict λ and minimize loss function). However, the neural network with four hidden layers shows better performance, with less error in predicting λ and minimizing the loss function after a certain number of iterations (30000). This neural

network is chosen for further analysis.

After 60000 iterations of training, the identified ODEs obtained by training the data with 1% noise or not are shown in the following Table 4.1. For training data without 1% nosie, the error in estimating λ_1 and λ_2 is 0.30699%, and 0.66179%, respectively. For training data with 1% nosie, the error in estimating λ_1 and λ_2 is 0.00799%, and 0.08091%, respectively.

Correct ODE	$\frac{du}{dx} + (x + \frac{1+3x^2}{1+x+x^3})u = x^3 + 2x + x^2\frac{1+3x^2}{1+x+x^3}$
Identified ODE (clean data)	$\frac{du}{dx} + (1.00x + \frac{1+3.00x^2}{1+x+x^3})u = x^3 + 2x + x^2\frac{1+3.00x^2}{1+x+x^3}$
Identified ODE (1% noise)	$\frac{du}{dx} + (1.00x + \frac{1+3.00x^2}{1+x+x^3})u = x^3 + 2x + x^2\frac{1+3.00x^2}{1+x+x^3}$

TABLE 4.1

Correct first-order linear ODE with $\lambda_1 = 1, \lambda_2 = 3$ along with the identified one obtained by learning λ for the cases of both noise-free data and 1% noise in the training data. The trained neural network has 8 hidden layers, 20 neurons in each hidden layer. An initial guess of $\boldsymbol{\lambda} = (0, 0)$ is provided.

Next, let's consider an example of a first-order non-linear ODE that can be represented by the following differential equation:

$$\frac{du}{dx} + xu^2 = 2\pi\cos(\pi x) + x\sin^2(2\pi x),\ u(0) = 0,\ x \in [0, 1]$$

Let's focus on solving this ODE with the initial value problem: $u(0) = 0$ for x in the interval $[0, 1]$ with the exact solution $u = sin(2\pi x)$. To do so, PINNs are introduced as an approach for solving this nonlinear ODE. PINNs involve training a neural network, denoted as $\Lambda(x; \theta)$, parameterized by θ, to approximate the solution $u(x)$ of the ODE.

In this example, 100 equally spaced points in the range $(0, 1)$ are generated for variables $\mathbf{x} = [x_1, x_2, \ldots, x_{100}]$ and $\mathbf{u} = [u_1, u_2, \ldots, u_{100}]$, where u_i is generated as $u_i = e^{(-x_i^2/2)/(1+x_i+x_i^3)} + x_i^2$ for $i = 1, 2, \ldots, 100$. A training set $\mathcal{T}_f$ of 50 pairs of points is randomly selected from $\mathbf{x}$ and $\mathbf{u}$ to enforce the differential equation $\mathcal{D}(x; \lambda)$, where $\mathcal{T}_f = (x_1, u_1), (x_2, u_2), \ldots, (x_{50}, u_{50})$. The differential equation $\mathcal{D}(x; \lambda)$ of the PINNs is defined as

$$\mathcal{D}(x; \lambda) \coloneqq \frac{du}{dx} + \lambda xu^2 = 2\pi\cos(\pi x) + x\sin^2(2\pi x).$$

The combined training set $\mathcal{T} = \mathcal{T}_b \cup \mathcal{T}_f$ is then used to train five different classes of neural networks, each with a different number of hidden layers (1, 2, 4, 7, and 10

hidden layers) with 20 neurons in each hidden layer. The performance of the trained neural network in predicting λ_1 and minimizing the loss function is shown in Figure B.2.

The first two plots show the error of the parameters λ are both close to 0 through the iteration, which means that through the iteration, the parameters are closer to the exact values. These five different classes of neural networks perform well (predict λ, and minimize loss function). However, the neural network has two hidden layers, shows better performance – less error of the predicted λ and minimizes loss function after a certain number of iterations (20000). Use two hidden layers neural network, and after 20000 iterations of training, the identified ODEs obtained by training the data with 1% noise or not are shown in Table 4.1. For training data without 1% noise, the error in estimating λ is 0.04377%. For training data with 1% noise, the error in estimating λ is 0.31911%.

Correct ODE	$\frac{du}{dx} + xu^2 = 2\pi\cos(2\pi x) + x\sin^2(2\pi x)$
Identified ODE (clean data)	$\frac{du}{dx} + 0.982xu^2 = 2\pi\cos(2\pi x) + x\sin^2(2\pi x)$
Identified ODE (1% noise)	$\frac{du}{dx} + 0.982xu^2 = 2\pi\cos(2\pi x) + x\sin^2(2\pi x)$

TABLE 4.2

Correct first-order non-linear ODE with $\lambda = 1$ along with the identified one obtained by learning λ for the cases of both noise-free data and 1% noise in the training data. The trained neural network has 8 hidden layers, 20 neurons in each hidden layer. An initial guess of $\lambda = 0$ is provided.

4.1.2. Second-order linear and non-linear ODEs. This section uses PINNs to solve the inverse problem for second-order linear and non-linear ODE. The analytic solution of ODEs and inverse parameters was known in advance.

As an example in second-order linear ODE, given the differential equation [9]

$$\frac{d^2u}{dx^2} + \lambda_1\frac{du}{dx} + \lambda_2 u = -\frac{1}{2}e^{-\frac{1}{2}x}\cos x, \tag{4.1}$$

consider the initial value problem: $u(0) = 1$ and $u(1) = e^{\frac{1}{2}}\sin 1$, with $x \in [0, 1]$. $\mathcal{T}_b = \{(0, 1), (1, e^{\frac{1}{2}}\sin 1)\}$ is defined as a training set for initial and boundary conditions. When $\lambda_1 = \frac{1}{2}$ and $\lambda_2 = 1$, the analytic solution of (4.1) is $u(x) = e^{-\frac{1}{2}x}\sin x$. The differential equation $\mathcal{D}(x; \lambda)$ of PINNs is defined as

$$\mathcal{D}(x; \lambda) \coloneqq \frac{d^2u}{dx^2} + \lambda_1\frac{du}{dx} + \lambda_2 u + \frac{1}{2}e^{-\frac{1}{2}x}\cos x,$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$. A neural network $\Lambda(x;\theta)$ with parameterized by θ approximates the solution $u(x)$. The loss function of the neural network $\Lambda(x;\theta)$ is defined by

$$\mathcal{L}(\theta, \lambda; \mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} \|u_i - \Lambda(x_i;\theta)\|^2 + \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{D}(x_i;\lambda)\|^2, \tag{4.2}$$

where $\mathcal{T} = \{(x_1, u_1), (x_2, u_2), \ldots, (x_N, u_N)\}$ is the training set.

In this example, 100 equally spaced points are generated in $(0, 1)$, which is $\mathbf{x} = [x_1, x_2, \ldots, x_{100}]$, and $\mathbf{u} = [u_1, u_2, \ldots, u_{100}]$, where $u_i = e^{-\frac{1}{2}x_i} \sin x_i$. 50 pair points $\mathbf{x}$ and $\mathbf{u}$ are choice randomly as training set $\mathcal{T}_f$ to enforce the differential equation $\mathcal{D}(x;\lambda)$, where $\mathcal{T}_f = \{(x_1, u_1), (x_2, u_2), \ldots, (x_{50}, u_{50})\}$. Using the training set $\mathcal{T} = \mathcal{T}_b \cup \mathcal{T}_f$ to train five different classes of neural networks, which have $1, 2, 4, 7, 10$ hidden layers, respectively, with 20 neurons in each hidden layer. The result is shown in Figure B.3. The first two plots show the error of the parameters (λ_1, λ_2) are both close to 0 through the iteration, which means that through the iteration, the parameters are closer to the exact values. The five different classes of neural networks perform well (predict (λ_1, λ_2), and minimize loss function). However, the neural network has four hidden layers, shows better performance – less error of the predicted λ and minimizes loss function after a certain number of iterations (10000). Using four hidden layers neural network, and after 20000 iterations of training, the identified ODEs obtained by training the data with 1% noise or not are shown in Table 4.3. For training data without 1% noise, the error in estimating λ_1 and λ_2 is 0.00292%, and 0.05392%, respectively. For training data with 1% noise, the error in estimating λ_1 and λ_2 is 0.10034%, and 4.45928%, respectively.

Correct ODE	$\frac{d^2u}{dx^2} + \frac{1}{2}\frac{du}{dx} + u = -\frac{1}{2}e^{-\frac{1}{2}x}\cos x$
Identified ODE (clean data)	$\frac{d^2u}{dx^2} + 0.501\frac{du}{dx} + 0.955u = -0.501e^{-0.501x}\cos x$
Identified ODE (1% noise)	$\frac{d^2u}{dx^2} + 0.501\frac{du}{dx} + 0.955u = -0.501e^{-0.501x}\cos x$

TABLE 4.3

Correct second-order linear ODE with $\lambda_1 = \frac{1}{2}, \lambda_2 = 1$ along with the identified one obtained by learning λ for the cases of both noise-free data and 1% noise in the training data. The trained neural network has 8 hidden layers, 20 neurons in each hidden layer. An initial guess of $\boldsymbol{\lambda} = (0, 0)$ is provided.

Next, consider an example in second-order non-linear ODE, given the differential

equation

$$\frac{d^2u}{dx^2} + \lambda_1 u^2 - \lambda_2 u = e^{\lambda_1 x}, \tag{4.3}$$

consider the initial value problem: $u(-1) = e^{-\frac{1}{2}}$ and $u(1) = e^{\frac{1}{2}}$, with $x \in [-1, 1]$. $\mathcal{T}_b = \{(-1, e^{-\frac{1}{2}}), (1, e^{\frac{1}{2}})\}$, defined as the training set for boundary condition. When $\lambda_1 = \frac{1}{2}$ and $\lambda_2 = 1$, the exact solution of (4.3) is $u(x) = e^{\frac{1}{2}x}$. The differential equation $\mathcal{D}(x; \lambda)$ of PINNs is defined as

$$\mathcal{D}(x; \lambda) \coloneqq \frac{d^2u}{dx^2} + \lambda_1 u^2 - \lambda_2 u - e^{\lambda_1 x},$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$. A neural network $\Lambda(x; \theta)$ with parameterized by θ approximates the solution $u(x)$.

In this example, 100 equally spaced points are generated in $(-1, 1)$, which is $\mathbf{x} = [x_1, x_2, \ldots, x_{100}]$, and $\mathbf{u} = [u_1, u_2, \ldots, u_{100}]$, where $u_i = e^{\frac{1}{2}x_i}$, for $i = 1, 2, \ldots, 100$. 50 pair points $\mathbf{x}$ and $\mathbf{u}$ are chosen randomly as the training set $\mathcal{T}_f$ to enforce the differential equation $\mathcal{D}(x; \lambda)$, where $\mathcal{T}_f = \{(x_1, u_1), (x_2, u_2), \ldots, (x_{50}, u_{50})\}$. Using the training set $\mathcal{T} = \mathcal{T}_b \cup \mathcal{T}_f$ to train five different classes of neural networks, which have $1, 2, 4, 7, 10$ hidden layers, respectively, with 20 neurons in each hidden layer. The result is shown in Figure B.4. The first two plots show the error of the parameters (λ_1, λ_2) are both close to 0 through the iteration, which means that through the iteration, the parameters are closer to the exact values. The five different classes of neural networks perform well (predict (λ_1, λ_2), and minimize loss function). However, the neural network that has two hidden layers, shows better performance – less error of the predicted λ and minimizes loss function after a certain number of iterations (20000). Use two hidden layers neural network, and after 30000 iterations of training, the identified ODEs obtained by training the data with 1% noise or not are shown in Table 4.4. For training data without 1% noise, the error in estimating λ_1 and λ_2 is 0.78785%, and 1.21614%, respectively. For training data with 1% noise, the error in estimating λ_1 and λ_2 is 0.03253%, and 0.03610%, respectively.

4.2. Two dimensional Laplace equation. In Sec 4.1, different inverse problems in ODEs were solved using PINNs. This section expands the inverse problem-solving by PINNs from ODEs to PDEs. Consider the following 2D Laplace equation

$$u_{xx} + \lambda u_{yy} = 0, \text{for} (x, y) \in [0, 5] \times [0, 3],$$

Correct ODE	$\frac{d^2u}{dx^2} + u^2 - \frac{1}{2}^2 u = e^{2\times\frac{1}{2}x}$
Identified ODE (clean data)	$\frac{d^2u}{dx^2} + 1.000u^2 - (0.500)^2 u = e^{2\times 0.500x}$
Identified ODE (1% noise)	$\frac{d^2u}{dx^2} + 1.000u^2 - (0.500)^2 u = e^{2\times 0.500x}$

TABLE 4.4

Correct second-order non-linear ODE with $\lambda_1 = 1, \lambda_2 = \frac{1}{2}$ along with the identified one obtained by learning λ for the cases of both noise-free data and 1% noise in the training data. The trained neural network has 8 hidden layers, 20 neurons in each hidden layer. An initial guess of $\boldsymbol{\lambda} = (0, 0)$ is provided.

and the boundary conditions

$$u(x = 0, y) = u(x = 5, y) = u(x, y = 0) = 0, u(x, y = 3) = 2. \tag{4.4}$$

The differential equation $\mathcal{D}(x, y; \lambda)$ of PINNs is defined as

$$\mathcal{D}(x, y; \lambda) \coloneqq u_{xx} + \lambda u_{yy}. \tag{4.5}$$

Aim to use known observation data and given boundary conditions to train a PINN to solve the parameter λ for the inverse problem. Raissi et al. [15] simulated using conventional spectral methods using the Chbfun package in MATLAB to obtain training and test data. This section obtains training data by solving a forward PDE problem using PINNs. Details of this are shown in Appendix A. For solving the above inverse PDE problem, $u(x, y)$ is approximated in (4.5) by a deep neural network $\Lambda(x, y; \theta, \lambda)$ parameterized by weights and biases in neural network and inverse parameter λ. The loss function of the neural network $\Lambda(x, y; \theta, \lambda)$ is defined by

$$\mathcal{L}(\theta, \lambda; \mathcal{T}) = \frac{1}{N}\sum_{i=1}^{N} \|u_i - \Lambda(x_i, y_i; \theta, \lambda)\|^2 + \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{D}(x_i, y_i; \lambda)\|^2, \tag{4.6}$$

where $\mathcal{T} = \{(x_1, y_1, u_1), (x_2, y_2, u_2), \ldots, (x_N, y_N, u_N)\}$ is the training set which not only for boundary conditions but also enforce the differential equation $\mathcal{D}(x, y; \lambda)$.

Here, 1000 pair points $(\mathbf{x}, \mathbf{y})$ are chosen randomly and corresponding $\mathbf{u}$ from the forward problem. After training, the neural network predicts the solution $u(x, y)$ and the unknown parameter λ for the inverse problem. The plot of the predictive solution $\hat{u}(x, y)$ is shown in Figure B.5 and B.6. The identified PDEs obtained by training the data with 1% noise or not are shown in Table 4.5. For training data without 1% noise, the error in estimating λ is 0.46864%. For training data with 1% noise, the error in estimating λ is 0.29508%.

Correct PDE	$u_{xx} + u_{yy} = 0$
Identified PDE (clean data)	$u_{xx} + 1.0047u_{yy} = 0$
Identified PDE (1% noise)	$u_{xx} + 1.0030u_{yy} = 0$

TABLE 4.5

Correct Laplace equation with $\lambda = 1$ along with the identified one obtained by learning λ for the cases of both noise-free data and 1% noise in the training data. The trained neural network has 8 hidden layers, 20 neurons in each hidden layer. An initial guess of $\lambda = 5$ is provided.

316 **5. Electrical Impedance Tomography.** Electrical Impedance Tomography
317 (EIT) is an imaging technique that uses electrical boundary measurements to re-
318 construct images with an object of interest [16]. Medical EIT uses these electrical
319 measurements to aid the diagnosis of certain medical conditions, including abnormal
320 tissue growths such as neoplasms [1], by reconstructing 2D or 3D images.

321 **5.1. Understanding the Functionality of EIT.** The report by CC. Chang
322 et al. [3] provides an illustration of the general layout and functionality of an EIT
323 system. The system described in this report involves placing a belt with 32 electrodes
324 around the patient's body. The electrodes work in pairs, with 2 electrodes emitting
325 a small electrical current and the other 30 electrodes measuring the resulting voltage
326 distribution. Each electrode in the pair measures and emits voltage in a loop, and
327 the resulting measurements are then sent to a signal amplifier. The amplifier filters,
328 digitizes and amplifies the data for visualization purposes. The resulting data is then
329 provided to an algorithm, which uses mathematical models to reconstruct an image of
330 the electrical conductivity distribution within the body. This image is then visualized
331 and analyzed by medical professionals for diagnosis and treatment purposes.

332 **5.2. EIT Benefits.** Unlike other medical imaging techniques, such as Comput-
333 erized Tomography (CT) or Magnetic resonance imaging (MRI), EIT is considered
334 non-invasive, low-cost and does not require the use of harmful ionizing radiation
335 [13]. One main advantage of EIT in a medical setting would be to monitor changes
336 in lung ventilation during mechanical ventilation or to detect pulmonary edema [5].
337 Whilst EIT has shown significant advancements, it does have certain limitations. The
338 foremost challenge is the ill-posed nature of the inverse problem, which entails recon-
339 structing internal electrical properties from the measurements on the boundary [10].
340 This results in multiple solutions that may fit the measured data with equal efficacy,

and determining the actual solution becomes difficult. EIT's limited resolution is another challenge, making it challenging to distinguish between adjacent tissues with similar electrical properties.

Moreover, EIT measurements are susceptible to noise and artifacts, leading to inaccurate reconstructed images. Recent advances in machine learning and neural networks have shown promise in addressing some of these challenges in medical EIT [17]. A potential solution to overcome the previously discussed limitations is using PINNs. This report explores the utilization of PINNs to resolve the medical EIT inverse problem and showcases their ability to enhance EIT image reconstruction.

5.3. Application in EIT. Consider the simple EIT problem, where the electric potential y is a measure of the voltage at each point in the tissue. The electrical conductivity of the tissue, denoted by σ, determines how easily electrical currents flow through it. The problem is defined over the domain $x \in [0, 1]$ and can be expressed by the following differential equation:

$$\frac{d}{dx}\left[\sigma(x)\frac{dy}{dx}\right] = 0.$$

To solve this problem using PINNs, the differential equation $\mathcal{D}(x, y; \lambda)$ is defined as follows

$$\mathcal{D}(x, y; \lambda) := \sigma(x) = e^{\lambda x}.$$

The goal is to obtain the training data by solving the forward problem using PINNs, and then use the given boundary conditions, $u(0) = 0$ and $u(1) = 0$, to train the model. To generate the training data, 52 equally spaced points (x, y) are sampled, with 50 points randomly generated in the domain $x \in [0, 1]$.

The first plot B.7 shows that the error of the parameter λ is close to 0% through the iteration, which means that through iteration, the parameter is equal to the exact value. In the second plot in B.7, the neural network converges faster with just one hidden layer since the loss function tends to zero the fastest. The five different neural network architectures all perform well (predict λ and minimize loss function). However, the neural network with one hidden layer shows better performance – less error of the predicted λ and minimizes loss function after a certain number of iterations. Therefore, using one hidden layer in the neural network yields the following identified

ODEs obtained by training the data with 1% noise or not shown in Table 5.1. Specifically, for training data without 1% noise, the error in estimating λ is 0.11058%. For training data with 1% noise, the error in estimating λ is 0.32052%.

Correct ODE	$\frac{d}{dx}(\sigma(x)\frac{dy}{dx}) = 0$	$\sigma(x) = e^{2x}$
Identified ODE (clean data)	$\frac{d}{dx}(\sigma(x)\frac{dy}{dx}) = 0$	$\sigma(x) = e^{1.998x}$
Identified ODE (1% noise)	$\frac{d}{dx}(\sigma(x)\frac{dy}{dx}) = 0$	$\sigma(x) = e^{2.006x}$

TABLE 5.1

Correct EIT equation along with the identified one obtained by learning λ for the cases of both noise-free data and 1% noise in the training data. The trained neural network has 1 hidden layer and 20 neurons in each hidden layer. An initial guess of $\lambda = 0$ is provided.

6. Conclusion and Future Research. In conclusion, this report has demonstrated the capabilities of PINNs in solving a diverse range of inverse problems. The examples presented in this report have illustrated the effectiveness of PINNs in accurately predicting the unknown parameters and solutions of both linear and non-linear ordinary and partial differential equations while being able to handle various types of boundary and initial conditions and noisy data.

The results presented in this report indicate that the performance of PINNs can be improved by optimizing the neural network architecture, selecting the optimal number of hidden layers and tuning hyper-parameters. By doing so, the accuracy of the network can be greatly enhanced.

Moreover, the analysis has highlighted that PINNs exhibit exceptional performance in EIT problems, with the capability to predict the unknown parameter with high accuracy. This represents a significant step forward in medical imaging and opens the door to new possibilities in diagnosing and treating various medical conditions.

In summary, it was found that there is compelling evidence that PINNs are a powerful tool in the field of inverse problems, with the potential to revolutionize the medical imaging industry.

As a potential avenue for future research, it would be worthwhile to explore and compare the computational efficiency and robustness of PINNs with other commonly used inverse problem-solving techniques, such as the Gauss-Newton method [7]. Further investigation could shed light on the comparative advantages and disadvantages of each method with respect to each problem.

Another avenue for future research would involve addressing the difficulties associated with using PINNs for 2D EIT forward problems, particularly when dealing with Neumann boundary conditions. Attempts to obtain an exact solution using only PINNs were unsuccessful, hindering the feasibility of conducting the reverse problem. To overcome this, it is suggested that alternative approaches or a combination of methods be explored to enhance the accuracy of the forward problem data, thus enabling more successful inverse modeling.

403 REFERENCES

404 [1] M. Akhtari-Zavare and L. A. Latiff, *Electrical impedance tomography as a primary screen-*
405 *ing technique for breast cancer detection*, Asian Pacific Journal of Cancer Prevention, 16
406 (2015), p. 5595–5597, https://doi.org/10.7314/apjcp.2015.16.14.5595.
407 [2] C. M. Bishop, *Neural networks and their applications*, Review of Scientific Instruments, 65
408 (1994), pp. 1803–1832, https://doi.org/10.1063/1.1144830.
409 [3] C. Chang, Z. Huang, S. Shih, H. Chen, J. Hsieh, and T. J. Kuo, *Electrical impedance*
410 *tomography for non-invasive identification of fatty liver infiltrate in overweight individ-*
411 *uals*, Scientific Reports, 11 (2021), p. 19859, https://doi.org/https://doi.org/10.1038/
412 s41598-021-99132-z.
413 [4] V. Dwivedi, N. Parashar, and B. Srinivasan, *Distributed learning machines for solving for-*
414 *ward and inverse problems in partial differential equations*, Neurocomputing, 420 (2021),
415 pp. 299–316, https://doi.org/https://doi.org/10.1016/j.neucom.2020.09.006.
416 [5] I. Frerichs, M. B. Amato, A. H. van Kaam, D. G. Tingay, Z. Zhao, B. Grychtol, M. Bo-
417 denstein, H. Gagnon, S. H. Böhm, E. Teschner, and et al., *Chest electrical impedance*
418 *tomography examination, data analysis, terminology, clinical use and recommendations:*
419 *Consensus statement of the translational eit development study group*, Thorax, 72 (2016),
420 p. 83–93, https://doi.org/10.1136/thoraxjnl-2016-208357.
421 [6] S. Goswami, C. Anitescu, S. Chakraborty, and T. Rabczuk, *Transfer learning enhanced*
422 *physics informed neural network for phase-field modeling of fracture*, Theoretical and Ap-
423 plied Fracture Mechanics, 106 (2020), p. 102447, https://doi.org/https://doi.org/10.1016/
424 j.tafmec.2019.102447.
425 [7] M. R. Islam and M. A. Kiber, *Electrical impedance tomography imaging using gauss-newton*
426 *algorithm*, 2014 International Conference on Informatics, Electronics amp; Vision (ICIEV),
427 (2014), https://doi.org/10.1109/iciev.2014.6850719.
428 [8] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, *Adaptive activation functions acceler-*
429 *ate convergence in deep and physics-informed neural networks*, Journal of Computational
430 Physics, 404 (2020), p. 109136, https://doi.org/https://doi.org/10.1016/j.jcp.2019.109136.
431 [9] I. Lagaris, A. Likas, and D. Fotiadis, *Artificial neural networks for solving ordinary and*
432 *partial differential equations*, IEEE Transactions on Neural Networks, 9 (1998), pp. 987–
433 1000, https://doi.org/10.1109/72.712178.
434 [10] W. R. Lionheart, *Eit reconstruction algorithms: Pitfalls, challenges and recent developments*,
435 Physiological Measurement, 25 (2004), p. 125–142, https://doi.org/10.1088/0967-3334/25/
436 1/021.
437 [11] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *Deepxde: A deep learning library for*
438 *solving differential equations*, SIAM Review, 63 (2021), pp. 208–228, https://doi.org/10.
439 1137/19M1274067.
440 [12] G. Montavon, W. Samek, and K.-R. Müller, *Methods for interpreting and understanding*
441 *deep neural networks*, Digital Signal Processing, 73 (2018), pp. 1–15, https://doi.org/10.
442 1016/j.dsp.2017.10.011.
443 [13] L. Parker, *Computed tomography scanning in children: Radiation risks*, Pediatric Hematology

and Oncology, 18 (2001), p. 307–308, https://doi.org/10.1080/088800101300312564.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in pytorch*, (2017), https://openreview.net/forum?id=BJJsrmfCZ.

[15] M. Raissi, P. Perdikaris, and G. 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), pp. 686–707, https://doi.org/https://doi.org/10.1016/j.jcp.2018.10.045.

[16] T. Strauss and T. Khan, *Implicit solutions of the electrical impedance tomography inverse problem in the continuous domain with deep neural networks*, Entropy, 25 (2023), 493, https://doi.org/10.3390/e25030493.

[17] T. Zhang, X. Tian, X. Liu, J. Ye, F. Fu, X. Shi, R. Liu, and C. Xu, *Advances of deep learning in electrical impedance tomography image reconstruction*, Frontiers in Bioengineering and Biotechnology, 10 (2022), https://doi.org/10.3389/fbioe.2022.1019531.

Appendices

A. Forward Problem for Laplace Equation. To show that PINNs also have high performance for solving forward PDEs problems, define $\lambda = 1$ in (4.2) and proceed by approximating $u(x, y)$ in (4.5) by a deep neural network $\Lambda(x, y; \theta)$ parameterized by θ. The loss function of the neural network $\Lambda(x, y; \theta)$ is defined by

$$\mathcal{L}(\theta; \mathcal{T}_b, \mathcal{T}_f) = \frac{1}{N_b} \sum_{i=1}^{N_b} \|u_i - \Lambda(x_i, y_i; \theta)\|^2 + \frac{1}{N_f} \sum_{i=1}^{N_f} \|\mathcal{D}(x_i, y_i)\|^2, \tag{A.1}$$

where $\mathcal{T}_b = \{(x_1, y_1, u_1), (x_2, y_2, u_2), \ldots, (x_{N_b}, y_{N_b}, u_{N_b})\}$ is the training set for initial and boundary conditions, and $\mathcal{T}_f = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{N_b}, y_{N_b})\}$ is the training set for enforce the differential equation $\mathcal{D}(x, y)$. 100 pairs points $(\mathbf{x}, \mathbf{y})$ are choice randomly and corresponding $\mathbf{u}$ in boundary conditions (4.4) to generate $\mathcal{T}_b$, and 1000 pairs points (x, y) in $[0, 5] \times [0, 3]$ to generate $\mathcal{T}_f$. Using hyperbolic tangent activation function and L-BFGS optimizer to train a neural network $\Lambda(x, y; \theta)$ with 8 hidden layers and 20 neurons in each hidden layer to minimize the loss function (A.1). The output of the neural network $\Lambda(x, y; \hat{\theta})$ after training is shown in Figure A.1.

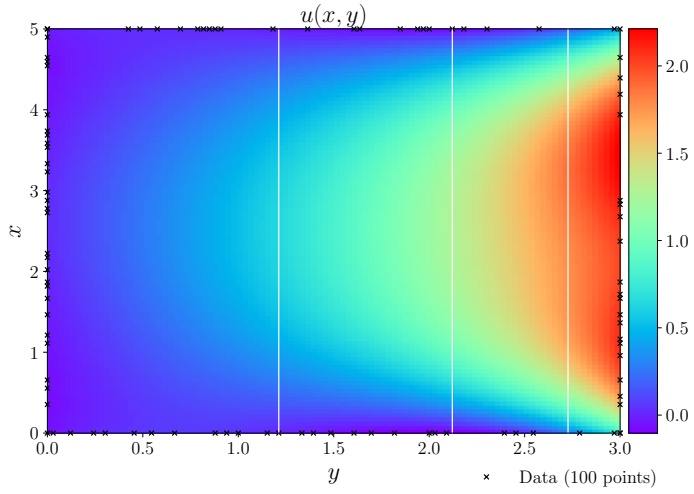

FIG. A.1. *Laplace Equation: Example for PDE forward problem. The plot shows the solution $\hat{u}(x, y)$ of the Laplace equation using the trained neural network, which has 8 hidden layers, 20 neurons in each hidden layer.*

B. Experiment Figures. This section shows all of the experiment figures mentioned above.

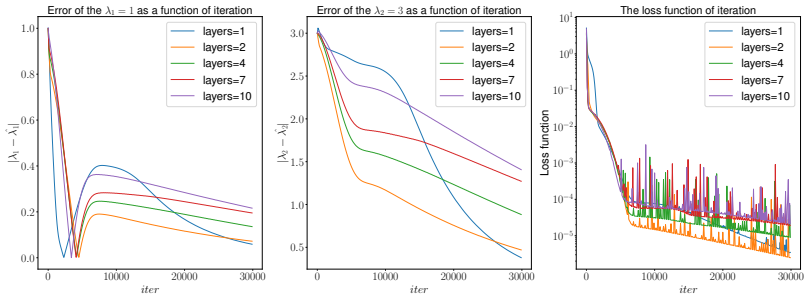

FIG. B.1. *Example for first-order linear ODE inverse problem. The first two plots show the error of the parameters* (λ_1, λ_2) *for five different neural networks through iteration. The third plot shows the value of the loss function through iteration.*

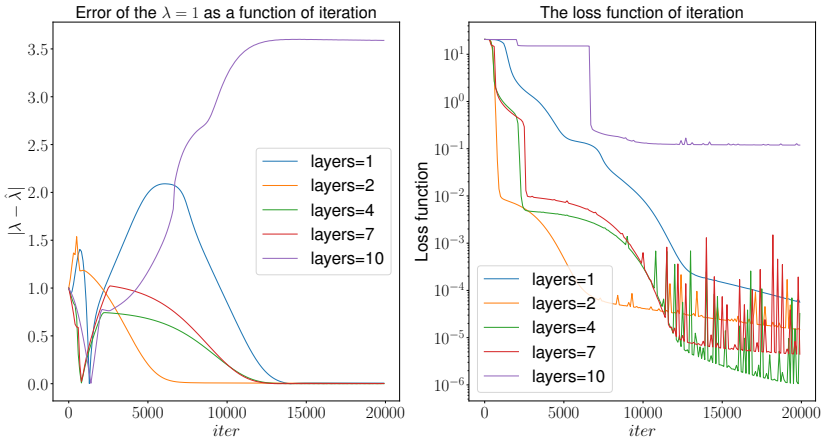

FIG. B.2. *Example for first-order non-linear ODE inverse problem. The first plot shows the error of the parameter* λ *for five different neural networks through iteration. The second plot shows the value of the loss function through iteration.*

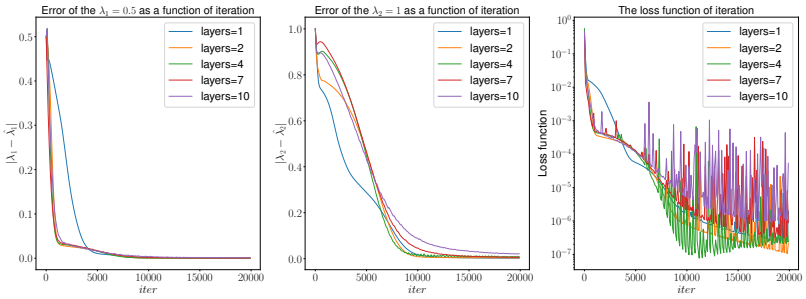

FIG. B.3. *Example for second-order linear ODE inverse problem. The first two plots show the error of the parameters* (λ_1, λ_2) *for five different neural networks through iteration. The third plot shows the value of the loss function through iteration.*

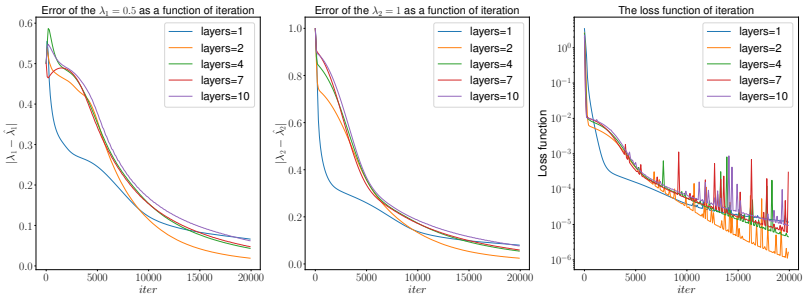

FIG. B.4. *Example for second-order non-linear ODE inverse problem. The first two plots show the error of the parameters* (λ_1, λ_2) *for five different neural networks through iteration. The third plot shows the value of the loss function through iteration.*

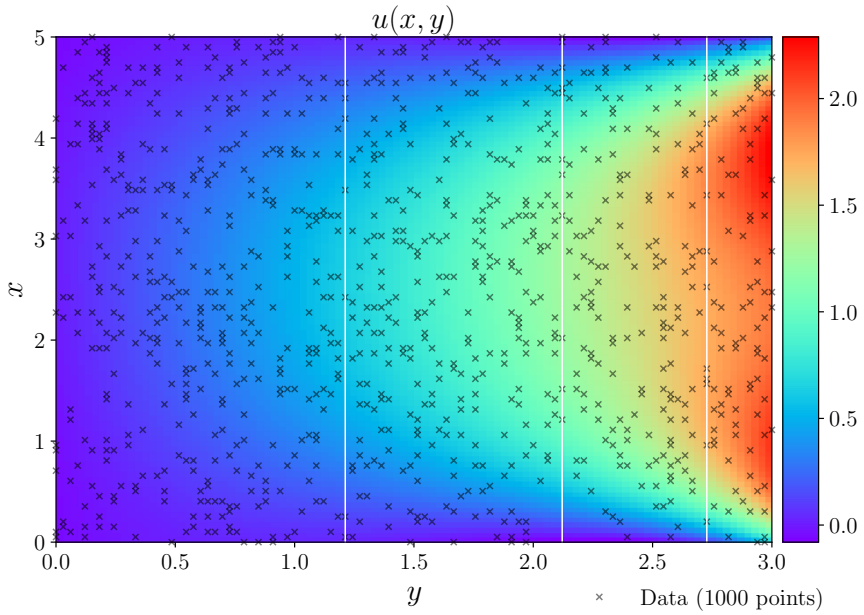

FIG. B.5. *Laplace Equation: Example for PDE inverse problem. The plot shows the solution* $\hat{u}(x, y)$ *of the Laplace equation using the trained neural network, which has eight hidden layers, 20 neurons in each hidden layer.*

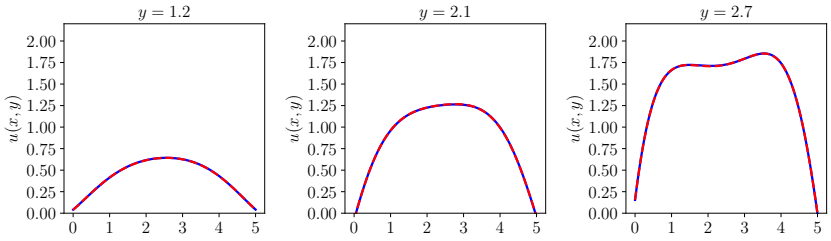

FIG. B.6. *Laplace Equation: Example for PDE inverse problem. Comparison of the predicted and exact solutions corresponding to the three case* $y = 1.2, y = 2.1$ *and* $y = 2.7$ *by the white vertical lines in Figure B.5.*

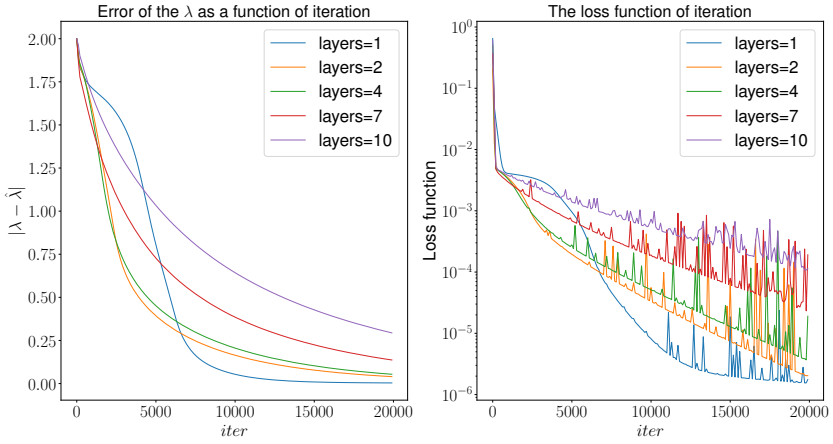

FIG. B.7. *EIT Equation: The first plot shows the error of the parameters* (λ_1, λ_2) *for five different neural networks through iteration. The second plot shows the value of the loss function through iteration.*