Physics and Equality Constrained Artificial Neural Networks: Application to Partial Differential Equations

Shamsulhaq Basir, Inanc Senocak

[a]*Department of Mechanical Engineering and Materials Science, University of Pittsburgh, Pittsburgh, 15261, PA, USA*

Abstract

Physics-informed neural networks (PINNs) have been proposed to learn the solution of partial differential equations (PDE). In PINNs, the residual form of the PDE of interest and its boundary conditions are lumped into a composite objective function as an unconstrained optimization problem, which is then used to train a deep feed-forward neural network. Here, we show that this specific way of formulating the objective function is the source of severe limitations in the PINN approach when applied to different kinds of PDEs. To address these limitations, we propose a versatile framework that can tackle both inverse and forward problems. The framework is adept at multi-fidelity data fusion and can seamlessly constrain the governing physics equations with proper initial and boundary conditions. The backbone of the proposed framework is a nonlinear, equality-constrained optimization problem formulation aimed at minimizing a loss functional, where an augmented Lagrangian method (ALM) is used to formally convert a constrained-optimization problem into an unconstrained-optimization problem. We implement the ALM within a stochastic, gradient-descent type training algorithm in a way that scrupulously focuses on meeting the constraints without sacrificing other loss terms. Additionally, as a modification of the original residual layers, we propose lean residual layers in our neural network architecture to address the so-called vanishing-gradient problem. We demonstrate the efficacy and versatility of our physics- and equality-constrained deep-learning framework by applying it to learn the solutions of various multi-dimensional PDEs, including a nonlinear inverse problem from the hydrology field with multi-fidelity data fusion. The results produced with our proposed model match exact solutions very closely for all the cases considered.

Keywords: Augmented Lagrangian Method, Constrained Optimization, Deep learning,

Multi-fidelity Data, Unconstrained Optimization, Helmholtz equation, Inverse Problems, Poisson's equation

1. Introduction

Machine learning has been highly impactful in a plethora of fields such as pattern recognition [1, 2], speech recognition [3], natural language processing [4, 5, 6] and in the solution of partial differential equations (PDE) for forward and inverse problems. The history of machine learning has been discussed in great depths in [7, 8]. It has been argued that the history of machine learning can be traced back to 300 BC, but the first primitive neural network was introduced by Warren McCulloch and Walter Pitts in 1943 [9].Years later, Rosenblatt [10] proposed the Perceptron that converges for linearly separable data. However, it was the seminal work of Hubel and Wiesel on the primary visual cortex of cats that had showed the existence of hierarchical layers of processing for visual stimulus, which has inspired modern neural networks [11]. Motivated by that innovation, the first deep neural network, termed the neocognitron, which incorporated neurophysiological insights, was introduced by Fukushima [12] in 1980, which came to be known as convolutional neural networks (CNN). The simplest form of a neural network is called a feed-forward neural network or multilayer perceptrons, which are parametric functions that learn to map a set of inputs to a set of outputs [13]. These networks can be specialized for certain tasks, such as CNN for object recognition, recurrent neural networks (RNN) for sequential data processing [14, 15, 16] and generative adversarial networks, or GANs for short, and autoencoders for unsupervised learning tasks [17, 18, 19, 20, 21].

The simplest neural-network model can be a parametric linear model, such as a linear regression model for representing linear functions. The parameters of this linear model can be discovered reliably either in closed form or with convex optimization methods. Convex optimization is a subfield of mathematical optimization dealing with optimization of convex functions with convex constraints where every local minimum is a global minimum [22]. Linear models, however, have limited capacities as they are only suitable to represent linear relations between input and output data. On the other hand, deep neural networks include stacks of parametric nonlinear transformations that can be optimized for a good representation of

the target nonlinear function [13]. Consequently, these nonlinearities can result in a non-convex objective function with respect to the parameters of the network. Hence, closed-form optimization of neural networks' parameters becomes impractical. Thus, iterative gradient-based algorithms are usually preferred for optimizing the parameters of neural networks. For gradient-based learning algorithms, the network parameters are randomly initialized [23, 24]. They are then adjusted to reduce the objective function. In neural networks, gradients of the objective function with respect to the parameters of the neural network are calculated using automatic differentiation [25] . Once the gradients are calculated, they can be fed to any optimization algorithm to minimize the objective function [26, 27, 13].

Development of the universal approximation theorem [28] has been a key milestone in the evolution of neural networks. According to this theorem a feed-forward network with as few as a single layer is sufficient to represent any function [28, 29]. This attribute of neural networks has sparked an interest in the scientific community to use neural networks to learn the solution of PDEs and ordinary differential equations (ODE) in the 90s [30, 31, 32, 33, 34]. Two of the earliest examples of using neural networks to solve PDEs are the works of Dissanayake and Phan-Thien [31] and van Milligen et al. [32]. We describe the method proposed in those works in more details in Section 2.1 as the overall framework proposed in those works is common in several variations of the modern physics-constrained learning approach. Dissanayake and Phan-Thien [31] used a feed-forward neural network with two hidden layers to solve a two-dimensional inhomogeneous Poisson's equation with a sinusoidal forcing term, where they lumped the residual form of the PDE and its boundary conditions into a composite objective function for training their neural network model. van Milligen et al. [32] used a similar approach to solve a two-dimensional magnetohydrodynamic plasma equilibrium problem. Lagaris et al. [33] used neural networks for the solution of ODEs and PDEs by creating custom trial functions that satisfied the boundary conditions automatically. The authors demonstrated their method by solving two-dimensional Poisson's equations with simple exponential and sinusoidal forcing terms. However, their work is not easily extensible to non-rectangular domains as it requires construction of auxiliary functions, which is only feasible for rectangular domains. Other researchers extended the approach of Dissanayake and Phan-Thien to the solution of nonlinear Schrodinger equation [35], chemical reactor

problems [36], self-gravitating systems of N bodies [37], and the solution of one-dimensional Burgers equation [38].

The overall technical approach for using neural networks to solve PDEs and ODEs that was adopted in the aforementioned works, particularly the method described in [31, 32], has found a resurgent interest in recent years [39, 40, 41, 42]. Raissi et al. [43] dubbed the term physics-informed neural networks (PINNs). They, too, formulated the loss function as a composite objective function using the residual forms of the PDE and its boundary conditions as mean squared error (MSE). The composite loss/objective function was then minimized using the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [44] to train a neural network and discover its parameters. Raissi et al. made use of the modern implementation of automatic differentiation techniques [25] in the popular deep-learning platform TensorFlow [45], whereas early researchers performed automatic differentiation manually partly because their networks were composed of a few hidden layers only. Raissi et al. [46] also demonstrated the potential of the PINN approach for fluid mechanics to learn the velocity and pressure fields in arbitrary domains by encoding the Navier-Stokes equations into their neural networks using passive concentration data at sparse locations. The physics-informed machine learning has been growing fast in popularity and applied to several unique forward and inverse problems [47, 48, 49, 50, 51, 52, 53, 54]. Extensive reviews of the current state in physics-informed machine learning are available in literature [55, 56].

One of the motivations for using neural networks in solving differential equations is their potential to break the curse of dimensionality [57, 58, 59]. Unlike the conventional mesh-based techniques such as finite difference, finite element, and volume methods, the meshless nature of neural networks also make them an attractive approach, albeit their slower performance compared to conventional numerical methods. Neural networks are advantageous for developing data-driven techniques for forward and inverse modeling problems as well [60]. In contrast to conventional numerical methods, once the model is trained, it can produce results at any point in the domain. In what follows, we present the PINN approach and identify a weak link in the construction of the loss function as an unconstrained optimization problem. We then formulate the machine learning of the solution of PDEs as a constrained-optimization problem in the first place and adopt an augmented Lagrangian method (ALM)

[61, 62] with a unique implementation to formally convert a constrained optimization problem into an unconstrained optimization problem. Additionally, we will present a modification of the residual layers networks to address the so-called vanishing gradient problem. We present several challenging numerical examples to demonstrate the efficacy and versatility of our approach over the original PINN approach.

2. Technical Background

In this section, we start with describing the PINN approach and discuss some of the technical issues arising from the unconstrained optimization approach adopted in PINNs. We then present the augmented-Lagrangian method [62, 61], an approach that has enjoyed a renewed interest in constrained-optimization problems [63].

2.1. Physics-informed Neural Networks

We focus our presentation on the common elements of the physics-informed learning framework that was presented in the works of Dissanayake and Phan-Thien [31] and van Milligen et al. [32], and, as part of contemporary developments in deep learning methods, in the work of Raissi et al. [43]. Consider a scalar function $u(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ on the domain $\Omega \subset \mathbb{R}^d$ with its boundary $\partial\Omega$ satisfying the following partial differential equation

$$\mathrm{D}(\boldsymbol{x}, u) = F(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{1}$$

$$\mathrm{B}(\boldsymbol{x}, u) = G(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega, \tag{2}$$

where D and B are differential operators on the interior and boundary respectively. F and G are source functions. The above PDE and its boundary conditions are then expressed in residual forms as follows:

$$\mathcal{D}(\boldsymbol{x}, \hat{u}) := \mathrm{D}(\boldsymbol{x}, \hat{u}) - F(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{3}$$

$$\mathcal{B}(\boldsymbol{x}, \hat{u}) := \mathrm{B}(\boldsymbol{x}, \hat{u}) - G(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega, \tag{4}$$

where $\hat{u}(\boldsymbol{x}, \theta)$ is an approximate solution in some underlying space $\mathcal{V}$, which satisfies Eqs. (1) and (2), and θ represents its parameters. Dissanayake and Phan-Thien proposed minimizing the sum of squared error (SSE) loss expressed as a composite objective function as follows:

$$\mathcal{L}(\theta) = \omega_\Omega \int_\Omega |\mathcal{D}(\boldsymbol{x}, \hat{u})|^2 dV + \omega_{\partial\Omega} \int_{\partial\Omega} |\mathcal{B}(\boldsymbol{x}, \hat{u})|^2 dS, \tag{5}$$

where the weights ω_Ω and $\omega_{\partial\Omega}$ were set to unity originally and the integration was approximated by sampling collocation points on Ω and $\partial\Omega$. In van Milligen et al., these weights were tuned manually. It is worth noting that the strong form of the partial differential equation is used in Eq. (5). For this purpose, the only explicit information that is needed to feed to the network is the boundary conditions. Therefore, this approach of learning the solution of PDEs while only feeding the boundary conditions can be viewed as a form of semi-supervised learning.

Finding the proper weights that appear in the composite objective function is of utmost importance as it helps balance the contributions to the composite objective function from Eqs. (3) and (4). Parisi et al. [36] used the above approach to solve an unsteady solid-gas reactor problem composed of three coupled first-order PDEs. Parisi et al. made these weights a function of the number of collocation points

$$\omega_{\partial\Omega} = \frac{1}{N_{\partial\Omega}}, \quad \omega_\Omega = \frac{1}{N_\Omega}, \tag{6}$$

where N_Ω and $N_{\partial\Omega}$ are the number collocation points in the domain and on the boundary, respectively. With the above change to the weights, the composite loss function given in Eq. (5) becomes equivalent to a sum of the individual mean of squared errors (MSE) resulting from the PDE and the boundary and initial conditions. In practice, Eq. (6) works better only because N_Ω is typically much larger than $N_{\partial\Omega}$, which essentially gives more weight to the boundary conditions in the loss function. Based on our own experience, we find that manual tuning of these weights is not ideal, because it creates a ripple effect as we then need to tune other hyperparameters, such as the number of collocations points, the learning rate, and the architecture.

Ad-hoc nature of determining these weights undermines the predictive capability of physics-informed learning in general. Alternative approaches to determining these weights have been proposed by other researchers. E and Yu [39] proposed to express the loss function in a variational form, where trial functions were represented by the neural networks and the boundary conditions on the PDE were used as a soft-penalty term in the loss function, but the penalty term included an ad-hoc parameter that was tuned for each problem. E and Yu acknowledged the non-convex nature of the variational formulation and potential difficulties in matching the boundary conditions. This variational form of the loss function

was also adopted in Zhu et al. [42]. van der Meer et al. [64] suggested a heuristic approach to approximate the weights in the loss function by casting the optimization problem as a multi-objective optimization problem with separate objectives arising from the solution of the PDE on the interior domain and satisfying the boundary conditions. The multi-objective optimization problem was then reduced to a single objective function through scalarization such that the loss function becomes a scaled version of the original form (i.e. Eq. 5). This normalized weight in the scaled loss function was then learned by a neural network using some form of the relative error. As stated by van der Meer et al., this approach is prone to learning a value from a local minimum, leading to poor predictions.

Wang et al. [65] also stressed the undesirable issues associated with manually tuning the weights in the composite objective function. Wang et al. proposed a heuristic learning rate annealing algorithm where the weights in the composite objective function were determined as a moving average of the back-propagated gradients of the loos function. Their suggestion resulted in improved accuracy for the solution of two dimensional Helmholtz equation, Klein-Gordon and the stream function vorticity formulation of the incompressible fluid flow equations with a relative L_2 norm of the error on the order of $10^{-2} - 10^{-3}$, whereas the L_∞ norm consistently exhibited an error on the order of 10^{-1}.

Formulation of the loss function as a composite objective function with tunable weights has been a common practice in recent applications of physics-informed learning, but other approaches to formulating the objective/loss function are possible and have been attempted in past studies. For instance, Lagaris et al. [66] considered a constrained-optimization approach and used a penalty method to constrain the PDE solution with the boundary conditions on the PDE. However, Lagaris et al. had to apply an extra network built with radial basis function on top of the first network with the penalty approach to enforce the boundary conditions on the PDE exactly. In Monterola and Saloma [67], the method of Lagrange multipliers was used to enforce constraints on characterizing the dynamics of physical systems, but the examples did not include constraining the solution of a PDE with boundary/initial conditions.

Constrained optimization is subfield of the optimization field in general and there are well-established methods for enforcing constraints of various forms. In the present work, we

consider the application of the so-called augmented-Lagrangian method (ALM), which was proposed independently by Powell [62] and Hestenes [61], to physics-constrained learning of PDEs. ALM is also called the *method of multipliers* and should not be confused with the *method of Lagrange multipliers*, although they are related. Modern applications of ALM are presented in Birgin and Martínez [63] in great details.

2.2. Augmented-Lagrangian Method for Constrained Optimization

Consider the following nonlinear, equality-constrained optimization problem with n decision variables, and m equality constraints

$$\begin{aligned} &\underset{\boldsymbol{\theta}\in\Theta}{\text{minimize}} \quad \mathcal{F}(\boldsymbol{\theta}), \\ &\text{subject to} \quad \boldsymbol{\mathcal{C}}(\boldsymbol{\theta}) = 0. \end{aligned} \tag{7}$$

where $\mathcal{F}$ is a nonlinear function of $\mathbb{R}^n$ in $\mathbb{R}$, $\boldsymbol{\mathcal{C}}$ is a nonlinear function of $\mathbb{R}^n$ in $\mathbb{R}^m$ and Θ is a given subset of $\mathbb{R}^n$, n-dimensional Euclidean space. There are mainly three classes of methods for solving Eq. (7). The penalty method [68] converts the constrained optimization problem of Eq.(7) into an unconstrained optimization problem by adding the constraints to the objective function as quadratic penalty terms as follows:

$$\underset{\theta\in\Theta}{\text{minimize}} \quad \mathcal{F}(\theta) + \frac{\mu^k}{2}\|\boldsymbol{\mathcal{C}}(\theta)\|_2^2, \tag{8}$$

for a scalar sequence $\{\mu^k\}$ such that $\mu^k \leq \mu^{k+1}$ for all k and $\mu^k \to \infty$. However, the main disadvantage of this method is that as $\mu \to \infty$ the penalty term prevails which induces ill-conditioning and makes the training unstable [68].

Another class of method for solving the constrained optimization problem of Eq. (7) is based on the sequential minimization of the Lagrangian function [68] defined for every θ in $\mathbb{R}^n$ and $\boldsymbol{\lambda}$ in $\mathbb{R}^m$ as follows:

$$\underset{\theta\in\Theta}{\text{minimize}} \quad \mathcal{L}(\theta, \boldsymbol{\lambda}) = \mathcal{F}(\theta) + \boldsymbol{\lambda}^T\boldsymbol{\mathcal{C}}(\theta), \tag{9}$$

where one minimizes the Lagrangian $\mathcal{L}(\theta, \boldsymbol{\lambda})$ over $\theta \in \Theta$ for a sequence of multiplier vectors $\{\boldsymbol{\lambda}^k\}$. This sequence is generated iteratively

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \alpha^k\boldsymbol{\mathcal{C}}(\theta^k), \tag{10}$$

where θ^k is the minimizing point of $\mathcal{L}(\theta, \boldsymbol{\lambda})$ over $\theta \in \Theta$, and α^k is the stepsize (scalar) parameter. The above formula is a steepest ascent iteration aimed at finding an optimal solution of an associated dual. For this reason, it is called a primal-dual method [22]. However, the disadvantage of this method is that the problem must have a locally convex structure and the Lagrangian must be minimized a large number of times since Eq. (10) converges moderately [68]. The final class of method for solving the constrained optimization problem of Eq. (7) is the augmented Lagrangian method (ALM), which was proposed independently by Powell [62] and Hestenes [61] and is also the method of choice in the present work. In ALM, the objective function is minimized possibly by violating the constraints. Subsequently, the feasibility is restored progressively as the iterations proceed [69]. ALM combines the desirable features of the penalty and the Lagrange multipliers methods for solving the constrained optimization problem of (7). In ALM, the penalty term is added not to the objective function $\mathcal{F}(\theta)$, but rather to its Lagrangian function, thus forming the "augmented" Lagrangian function as follows:

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \mathcal{L}_{\mu^k}(\theta, \lambda) = \mathcal{F}(\theta) + \boldsymbol{\lambda}^T \boldsymbol{\mathcal{C}}(\theta) + \frac{\mu^k}{2} \|\boldsymbol{\mathcal{C}}(\theta)\|_2^2, \tag{11}$$

where $\{\mu^k\}$ is a sequence of positive penalty parameters. Minimization of Eq. (11) can be performed with any variant of gradient descent methods [70, 26]. The multiplier sequence is generated from the following update rule

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mu^k \boldsymbol{\mathcal{C}}(\theta^k). \tag{12}$$

An important aspect of the ALM is that convergence of the method may be induced by not just increasing the penalty parameter μ, but also invoking multiplier iterations as in Eq.(12). As a result, convergence in ALM may occur with finite μ^k, effectively removing the ill-conditioning issue that arises in the conventional penalty method. Furthermore, the multiplier iteration (12) converges to a Lagrange multiplier vector much faster than the iteration of the primal-dual method (Eq. 10), and the optimization problem does not even have to possess a locally convex structure [68]. These aspects of the ALM make it a suitable choice for neural networks as their objective functions are typically non-convex with respect to the parameters of the network.

3. Proposed Method: Physics & Equality Constrained Artificial Neural Networks

Thus far, we have discussed the current state in PINNs and explained the overall framework in sufficient details to be able to elaborate on potential areas for improvement. Here, we present our contributions to the current state in physics-informed (constrained) machine learning. Our first contribution is to the way the optimization of the network parameters is formulated in the first place. We propose to cast the minimization of the residual form of the PDE as a constrained optimization problem using the boundary conditions on the PDE. To this end, we use the augmented Lagrangian method [62, 61] and implement it in a unique way to make it work for constraining a PDE solution. The principled application of the ALM eliminates the need for manual tuning of weights in the loss function and produces much more improved predictions as we show in later sections. Our second contribution to physics constrained learning is addressing the infamous problem of vanishing gradients by adapting the original residual layers, which are the building blocks for residual neural networks (ResNets) [2]. We find that the straightforward application of the ResNets for learning a PDE is not optimal. Consequently, we modify the original residual layers in a unique way to make them learn PDEs with better accuracy. Together, these two contributions comprise, what we call, the physics and equality constrained artificial neural networks (PECANN, for short).

3.1. Objective Function Formulation

Our goal is to constrain the residual form of the PDE with its boundary and initial conditions and, should they exist, with multi-fidelity observed data in a principled fashion. Therefore, our focus is on enforcing equality constraints in an optimization problem. We should stress that inequality constraints can be handled within ALM, should they exist for a problem of interest [63]. An example of enforcing inequality constraints is presented in Lu et al. [71], where they constrained an inverse design, topology optimization problem with PDE solutions using ALM.

Definition 1. *Distance function* $\mathcal{Q}(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ *is a quadratic function that is acting element-wise on its input vector* $\boldsymbol{t}$ *such that*

$$\mathcal{Q}(\boldsymbol{t}) = \boldsymbol{t} \odot \boldsymbol{t} \geq 0,$$

where $\odot$ is an element-wise multiplication operator. Before we discuss our neural networks architecture, we construct a general objective function that can admit low-fidelity measurement data, high-fidelity measurement data, boundary conditions, initial conditions, and governing equations for the physics of the problem at hand. For notation clarification, we stress that $\tilde{(\cdot)}$ and $\bar{(\cdot)}$ represent low fidelity and high fidelity data respectively. Given a set of low-fidelity measurement data $\{(\tilde{\boldsymbol{x}}^{(1)}, \tilde{u}^{(1)}), \cdots, (\tilde{\boldsymbol{x}}^{(N)}, \tilde{u}^{(N)})\}$ and a set of high-fidelity measurement data $\{(\bar{\boldsymbol{x}}^{(1)}, \bar{u}^{(1)}), \cdots, (\bar{\boldsymbol{x}}^{(M)}, \bar{u}^{(M)})\}$ along with the residual form of a general PDE Eq.(3) constrained by its boundary conditions Eq. (4), we can construct the following unconstrained objective function,

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \mathcal{L}_\mu(\theta, \boldsymbol{\lambda}) = \boldsymbol{\mathcal{D}}^T\boldsymbol{\mathcal{D}} + \tilde{\boldsymbol{\mathcal{D}}}^T\tilde{\boldsymbol{\mathcal{D}}} + \bar{\boldsymbol{\lambda}}^T\mathcal{Q}(\bar{\boldsymbol{\mathcal{D}}}) + \hat{\boldsymbol{\lambda}}^T\mathcal{Q}(\boldsymbol{\mathcal{B}}) + \frac{\mu}{2}\sum \mathcal{Q}(\bar{\boldsymbol{\mathcal{D}}})^T\mathcal{Q}(\bar{\boldsymbol{\mathcal{D}}}) + \mathcal{Q}(\boldsymbol{\mathcal{B}})^T\mathcal{Q}(\boldsymbol{\mathcal{B}}), \tag{13a}$$

$$\hat{\boldsymbol{\lambda}} = \hat{\boldsymbol{\lambda}} + \mu\mathcal{Q}(\boldsymbol{\mathcal{B}}), \tag{13b}$$

$$\bar{\boldsymbol{\lambda}} = \bar{\boldsymbol{\lambda}} + \mu\mathcal{Q}(\bar{\boldsymbol{\mathcal{D}}}), \tag{13c}$$

where μ is the penalty parameter, $\boldsymbol{\mathcal{D}}$ denotes an N_Ω-vector $\{\mathcal{D}(\boldsymbol{x}^{(1)}, \hat{u}^{(1)}), \cdots, \mathcal{D}(\boldsymbol{x}^{(N_\Omega)}, \hat{u}^{(N_\Omega)})\}$ viewed as a column vector containing the outputs of the collocation points calculated on the residual form of the PDE as in Eq. (3), and T denotes transposition. Likewise, $\boldsymbol{\mathcal{B}}$ denotes an $N_{\partial\Omega}$-vector $\{\mathcal{B}(\boldsymbol{x}^{(1)}, \hat{u}^{(1)}), \cdots, \mathcal{B}(\boldsymbol{x}^{(N_{\partial\Omega})}, \hat{u}^{(N_{\partial\Omega})})\}$ viewed as a column vector containing the outputs of the collocation points calculated on the boundary operator as in Eq. (4). In addition, $\tilde{\boldsymbol{D}}$ represents an N-vector $\{(\hat{u}^{(1)} - \tilde{u}^{(1)}), \cdots, (\hat{u}^{(N)} - \tilde{u}^{(N)})\}$ divergence between noisy measurement data and the model predictions on its collocation points viewed as a column vector. Also, $\bar{\boldsymbol{D}}$ represents an M-vector $\{(\hat{u}^{(1)} - \bar{u}^{(1)}), \cdots, (\hat{u}^{(N)} - \bar{u}^{(M)})\}$ divergence between high fidelity data and the model predictions on its collocation points viewed as a column vector. Finally, $\bar{\boldsymbol{\lambda}}$ and $\hat{\boldsymbol{\lambda}}$ denote the vectors of multipliers for the clean data and the boundary conditions, respectively. Therefore, any equality constraints, should they arise, can be incorporated in a similar fashion to the objective function $\mathcal{L}$ as in Eq.(13a). We can reduce our objective function into the canonical form of ALM by introducing the following

parameters,

$$\mathcal{F} = \boldsymbol{\mathcal{D}}^T \boldsymbol{\mathcal{D}} + \tilde{\boldsymbol{\mathcal{D}}}^T \tilde{\boldsymbol{\mathcal{D}}}, \tag{14a}$$

$$\boldsymbol{\lambda} = [\hat{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}}], \tag{14b}$$

$$\boldsymbol{C} = [\mathcal{Q}(\bar{\boldsymbol{\mathcal{D}}}), \mathcal{Q}(\boldsymbol{\mathcal{B}})]. \tag{14c}$$

The particular form of the distance function $\mathcal{Q}$ deserves a discussion. We use a quadratic function to calibrate the violation in our constraints. We choose this distance function intentionally for multiple reasons. To begin with, we avoid the issue of residual cancellation due to differing signs since the loss is calculated over a set of training points in machine learning applications. Otherwise, the accumulated errors might end up being either minuscule or cancel out to the point where the network has not learned anything but the user might think it has. Also, a quadratic function is differentiable. We note that one could also take the absolute value function, but this function is not differentiable and its gradient is constant with respect to its inputs. In other words, its gradient for a large error and a small error is the same. Another option would be to choose a fourth-order function or quartic function but the gradient of this function near the origin is flat which causes stiff learning. Therefore, a quadratic function is desirable for neural networks in the ALM context. Now that we have the formulation down, it is necessary to apply it under the stochastic training setting of neural networks.

3.2. Training Algorithm

There are mainly three variants of gradient descent algorithms for training neural networks [13, 26]. Batch gradient descent (BGD) processes the entire training set simultaneously. However, gradient calculations can be slow and impractical for the training sets that cannot fit in the memory [26]. On the other hand, stochastic gradient descent (SGD) technique updates the network parameters using a single training example at a time. While SGD allows using large training sets and makes frequent updates to the network parameters, the objective function heavily fluctuates as the updates are based on the gradient of a single training example. Most machine learning algorithms are known as mini-batch gradient descent methods that enjoy the merits of both BGD and SGD methods. Mini-batch gradient descents are known to be efficient in utilizing multi-core architectures [13].

Algorithm 1: Mini-batch augmented-Lagrangian based neural networks training algorithm

Input: $\theta^0, \mu^\infty, E, S$

/* Initializing the multipliers */

$\lambda_i \leftarrow 0, \forall i = 1, .., m$

/* Assigning the tolerance for constraints violation */

$\epsilon \leftarrow 10^{-8}$

/* Initializing the penalty term */

$\mu^0 \leftarrow 1.0$

/* Placeholder for violation of constraint */

$\eta \leftarrow 0.01$

Output: θ^*

for $epoch \leftarrow 1$ **to** E **do**

  /* Iterate over all training batches */

  for $batch \leftarrow 1$ **to** S **do**

    /* minimize the augmented Lagrangian with a stochastic gradient-descent technique */

    $\theta^* \leftarrow \underset{\theta}{\operatorname{argmin}} \quad \mathcal{F}(\theta) + \lambda^T \boldsymbol{C}(\theta) + \frac{\mu}{2} \|\boldsymbol{C}(\theta)\|_2^2$

    if $(\|\boldsymbol{C}(\theta)\|_2 \geq 0.25\eta)$ & $(\|\boldsymbol{C}(\theta)\|_2 > \epsilon)$ **then**

      /* Updating the penalty term */

      $\mu = \min(2\mu, \mu^\infty)$

      /* Updating the multiplier */

      $\boldsymbol{\lambda} = \boldsymbol{\lambda} + \mu \boldsymbol{C}(\theta)$

    /* Recording the current penalty loss */

    $\eta = \|\boldsymbol{C}(\theta)\|_2$

  end

end

In Algorithm 1, we present a training algorithm using the objective function presented in (13a). The input to the algorithm is an initialized set of parameters for the network, a maximum value μ^{∞} for safeguarding the penalty term, the number of epochs E, and the training set S. We should note that over-focusing on the constraints might result in a trivial prediction, where the constraints are satisfied, but the solution has not been found. Therefore, we tackle this issue by updating the multipliers when two conditions are met simultaneously: First, the ratio of the penalty loss term from successive iterations has not decreased. Second, the maximum allowable violation on the constraints have not been met. The first condition helps prevent aggressive updating of multipliers that might cause the aforementioned issue. In the second condition, we basically relax updating the multipliers if a satisfactory precision set by the user ϵ has been achieved. This, in turn, enables the network to freely choose to optimize any loss terms in the objective function so as to not sacrifice any loss term.

3.3. Neural Networks Architecture

In this section, we introduce our neural networks architecture that we use in both the PINN and our proposed PECANN models. We start by describing a conventional layer that has a weight layer and a nonlinear activation function as presented in Fig. 1(a). For conventional feed-forward neural networks, these nonlinear activations may saturate and cause the notoriously known vanishing-gradient problem, which makes learning significantly stiff. He et al. [2] proposed residual learning to address the performance degradation of deep convolutional networks (CNN) caused by the vanishing gradient problem. CNNs are designed for object recognition tasks. We should also note that the authors used extremely deep networks with up to 1000 layers in their study as such deep networks are not used in our current study.

He et al. used a shortcut connection to explicitly force a stack of nonlinear layers $\mathcal{G}$ to learn a residual mapping $\mathcal{R} := \mathcal{G} + x$, as shown in Fig.1(b). This shortcut connection perform identity mapping and their output is added to the output of the nonlinear layers, which is sent through another nonlinear σ transformation. In the original introduction of residual layers this non-linearity function was the dying $relu$. However, for our applications we adopt tangent hyperbolic ($tanh$) actitvation function because it is symmetric and many

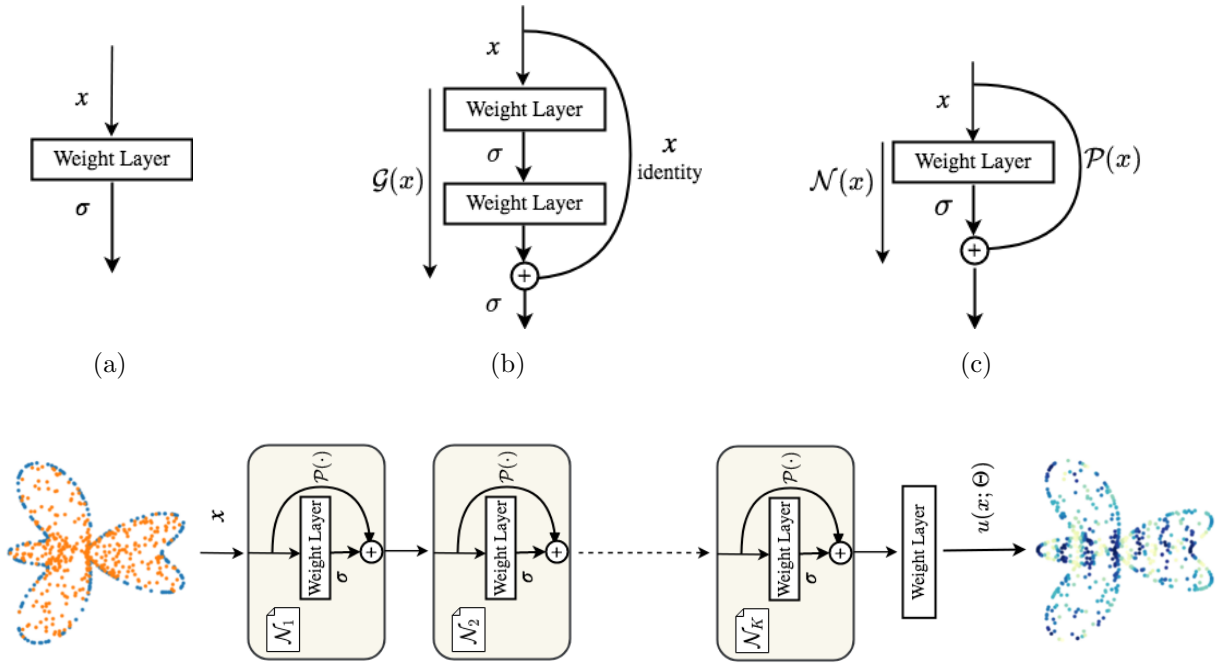

Figure 1: Schematic representation of the neural networks adopted in the present study: (a) a single layer of a conventional neural network with a nonlinear activation function denoted as σ, (b) a single unit of the original residual layer that represents a nonlinear function $\mathcal{G}$ with an nonlinear activation function σ, (c) proposed modified residual layer that represents a nonlinear function $\mathcal{N}$ with a skip-connection operator $\mathcal{P}$, (d) proposed neural network architecture with K modified residual layers $\mathcal{N}$ and skip-connection operators $\mathcal{P}$ with a set of parameters Θ and nonlinear activation functions σ.

times differentiable, as the governing laws incorporated into our objective functions require taking multiple derivatives of the output of the network with respect to the input of the network. We note that sending the output $\mathcal{R}$ through another non-linearity after $\bigoplus$ might still cause the vanishing gradient problem as pointed by the authors in [72]. Therefore, we remove this non-linearity for two purposes: First, we reduce the risk of vanishing gradient to reoccur by not having this nonlinearity function, which consequently allows direct propagation of information not just in the residual layer but throughout the entire network of residual layers. Second,it allows us to make use of the already learned features from the bottom layers and to connect each layer to every other upper layers. Having removed this last nonlinear activation function after the summation junction $\bigoplus$, we observe that the weight

layer before the junction becomes redundant because the output of the current residual layer will be fed to another residual layer that processes its input through a weight layer. In other words, linearly stacking two weight layers can be collapsed into a single weight layer. Therefore, we eliminate this extra weight layer to have a modified, leaner residual layer that has no extra weight or activation function in comparison to its conventional counterpart. A schematic representation of our proposed modified residual layer is shown in Fig. 1(c) with $\mathcal{P}(x)$ mappings. When the input dimension d and the number of neurons n in the weight layer are different, we use a linear mapping $\mathcal{P}(\boldsymbol{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^n$ to project the inputs to the correct dimension for component-wise addition purpose $\bigoplus$ in the residual layer, otherwise it is an identity mapping. Therefore,we have a leaner modified residual layer that is robust against the vanishing gradient problem. For further clarification of this behavior, we proffer a simple mathematical description of this phenomenon for a single modified residual layer with output value u and an input value of x. Consider a single modified residual layer as in Fig. 1(c) with a loss value denoted as $\mathcal{J}$ and an identity mapping $\mathcal{P}(x) = x$. We can mathematically demonstrate that the gradient of the loss $\mathcal{J}$ with respect to the input x does not die out even if the activation function σ in the nonlinear part of the residual layer $\mathcal{G}$ saturates

$$\frac{\partial \mathcal{J}}{\partial x} = \frac{\partial \mathcal{J}}{\partial u}(\frac{\partial u}{\partial \mathcal{N}}\frac{\partial \mathcal{N}}{\partial x} + \frac{\partial u}{\partial x}) = \frac{\partial \mathcal{J}}{\partial u}(\frac{\partial u}{\partial \mathcal{N}}\frac{\partial \mathcal{N}}{\partial x} + 1). \tag{15}$$

From Eq.(15), we observe that the gradient of loss with respect to input $\frac{\partial \mathcal{J}}{\partial x}$ will not vanish even if $\frac{\partial \mathcal{N}}{\partial x} \rightarrow 0$. To gain additional insight into our proposed residual neural network, we explore the behavior of a three-layer residual network with modified residual layers. Consider $\mathcal{N}_1$, $\mathcal{N}_2$,and $\mathcal{N}_3$ representing the first, the second and the third modified residual layers. We can expand the mappings from the input layer x to the output layer as follows:

$$x :\rightarrow [\mathcal{N}_1(x), \mathcal{N}_2(\mathcal{P}(x) + \mathcal{N}_1(x)), \mathcal{N}_3(\mathcal{P}(x) + \mathcal{N}_1(x) + \mathcal{N}_2(\mathcal{P}(x) + \mathcal{N}_1(x)))]. \tag{16}$$

We observe that the third layer $\mathcal{N}_3$ is not only a function $\mathcal{N}_2$, but also a function of $\mathcal{N}_1$ and $\mathcal{P}(x)$. Consequently, we write an output representation of a modified residual network with K residual layers and one output weight layer with parameters θ_L as follows:

$$u(\boldsymbol{x}; \Theta) = \theta_L^T[\mathcal{N}_K +, \cdots, + \mathcal{N}_1 + \mathcal{P}(x)], \tag{17}$$

where θ_L represents another vector of $n-$dimensional Euclidean space and T represents transposition. We observe from Eq.(17) that our network elegantly maps increasingly complex functions and aggregate them with proper weights to make a final prediction. We will demonstrate the performance gains and efficiency of our architecture with modified residual layers over another architecture with feed-forward neural networks with conventional layers as well as a network with the original residual layers on a pedagogical example presented in our results section. Fig.1(d) presents a schematic representation of a neural network architecture with K modified residual layers.

3.4. Training Details

In this section, we present the hyperparameters for efficiently training our networks. We choose Adam[70] with its initial learning rate set to 10^{-2} as our optimizer because it is adapting the learning rates for each parameter in the network during training. We use ReduceLROnPlateau(patience=100,factor=0.95) that is built-in Pytorch[73] framework to dynamically reduce the learning rate during training. We choose the hyperbolic tangent (tanh) function as the nonlinear activation function in all the networks considered in this work. The network parameters are initialized with the Kaiming initialization technique [24], unless otherwise stated.

The architecture for all the problems considered is composed of five modified residual layers with 50 neurons per layer. The networks are trained for 25000 epochs in all the problems. As for the training set, we uniformly generate 400 collocation points on the domain with 64×4 boundary points for two-dimensional problems and 64×6 boundary points for three-dimensional problems at every epoch.

As mentioned in the preliminary section for ALM, μ^{∞} can be a finite number since it does not have to approach to infinity for ALM to converge. We also justified this attribute of ALM with a pedagogical example by investigating the dependency and sensitivity of our model predictions on the limiting value of the penalty parameter μ^{∞}. We also provide a heuristic approach to setting μ^{∞}. First, we assume that we update our multipliers $\boldsymbol{\lambda}$ at every mini-batch and every epoch, for a given number of epochs E and a number of mini-batches $|S|$. Also, assuming that $|\boldsymbol{\mathcal{C}_i}| \to \mathcal{O}(1)$ for all i, we show that the update rule for the multipliers

follow a geometric sequence as follows:

$$1 + 2 + 2^2 + 2^3 +, \cdots, +2^n + E|S|2^n = \left(\frac{1 - 2^{n+1}}{1 - 2}\right) + E|S|2^n, \tag{18}$$

where μ^∞ corresponds to the value of 2^n in the above equation. The limiting value of our multipliers will approach the sum of the above series. Assuming $|\boldsymbol{\lambda}_i| \to \mathcal{O}(10^5)$ and $|\boldsymbol{\lambda}_i \boldsymbol{\mathcal{C}_i}| \to \mathcal{O}(10^{-3})$ for all i then it is easy to see that $\boldsymbol{\mathcal{C}_i} \to \mathcal{O}(10^{-8})$. In this work, we set the limiting penalty parameter $\mu^\infty = 500$, unless otherwise stated. All the codes in this work are developed in PyTorch framework [73] and will be publicly available on https://github.com/HiPerSimLab/PECANN.

4. Numerical Experiments

We provide several PDE problems with increasing complexity to demonstrate the efficacy of our proposed PECANN model and compare its accuracy relative to the PINN approach. Two additional examples, one for a three-dimensional problem and another for a time-dependent problem are included in Appendix A. To establish a fair comparison between the PINN and PECANN models, we use the same network architecture for both approaches under the same training settings. We use the method of manufactured solutions (MMS) [74] to generate exact solutions for several PDEs, due in part that complex solutions can easily be generated. First, we demonstrate implementation details of our proposed method by solving a pedagogical one-dimensional Poisson's equation. The objective here is to show a detailed analysis of our method and justify the use of our modified residual network. We then use the proposed residual network for the rest of the problems reported in this work. We solve several multi-dimensional Poission's equations, because these equations have been challenging to learn with neural networks as elliptic equations lack any characteristic path which makes the solution at every point in the domain be influenced by all the other points. Therefore, enforcing the boundary conditions correctly becomes more important when learning the solution of elliptic equations. We also demonstrate the performance of our method by solving a two-dimensional Poisson equation on a complex domain to highlight its applicability for PDEs on irregular domains. The two-dimensional time-dependent nonlinear Klein-Gordon equation example that we present in Appendix A showcases the generalization

of our approach to different boundary conditions and different types of PDEs. In addition to aforementioned forward problems, we tackle an inverse modeling problem where we learn the parameters of a one-dimensional unsaturated soil column with variable water content.

In the forward model examples, we assess the accuracy of our results by providing the L_∞ and the relative $L_{2,r}$ error separately on the domain and on the boundaries for both the PINN and PECANN models. Given two sets of vectors $\hat{\boldsymbol{u}}$, the learned solution, and $\boldsymbol{u}$, the exact solution, we define the relative $L_{2,r}$ of $\hat{\boldsymbol{u}}$ as follows,

$$L_{2,r} = \frac{\|\hat{\boldsymbol{u}} - \boldsymbol{u}\|_2}{\|\boldsymbol{u}\|_2}. \tag{19}$$

We should note that the relative $L_{2,r}$ error is a percentage accuracy measure of the model and it is different than the L_2 norm used in mesh-based numerical methods to assess the order of accuracy of a numerical solution. In Table 1, we present a summary of the L_∞ and $L_{2,r}$ norm of the errors resulting from applying the PINN and PECANN models to all the forward modeling problems considered in this study. The errors are broken into errors generated from the PDE domain and from the boundary. Comparison of the errors resulting from the application of the PINN and the PECANN models shows that there is a significant difference in the accuracy levels between these two approaches in favor of the PECANN model, despite adopting the same neural network architectures (i.e. modified residual layers with same training details). We observe that L_2 and L_∞ error levels from the PECANN model are consistently two to three orders of magnitude lower than the error levels from the PINN model.

4.1. Pedagogical Example

The aims of this pedagogical example are twofold: First, we thoroughly demonstrate the implementation intricacies of our proposed method and highlight its advantages over the PINN approach. Second, we demonstrate the significant improvement achieved in the model prediction using our modified residual architecture relative to the original residual networks [2].

Table 1: Error levels resulting from the application of the PINN and PECANN models to all the forward modeling problems considered in the present study. Errors are calculated separately for the solution domain Ω and the boundary $\partial\Omega$. Relative $L_{2,r}$ norm of error is calculated according to Eq. (19).

Simulation Problem	PINN				PECANN			
	$\boldsymbol{L_{2,r} \in \Omega}$	$\boldsymbol{L_\infty \in \Omega}$	$\boldsymbol{L_{2,r} \in \partial\Omega}$	$\boldsymbol{L_\infty \in \partial\Omega}$	$\boldsymbol{L_{2,r} \in \Omega}$	$\boldsymbol{L_\infty \in \Omega}$	$\boldsymbol{L_{2,r} \in \partial\Omega}$	$\boldsymbol{L_\infty \in \partial\Omega}$
1D Poisson's	4.649×10^{-2}	5.741×10^{-2}	5.705×10^{-2}	5.741×10^{-2}	1.343×10^{-4}	1.643×10^{-4}	2.218×10^{-5}	3.136×10^{-5}
2D Poisson's	5.052×10^{-1}	6.343×10^{-1}	5.763×10^{-1}	6.344×10^{-1}	2.623×10^{-3}	4.859×10^{-3}	2.195×10^{-3}	4.863×10^{-3}
2D Helmholtz	3.510×10^{-1}	5.984×10^{-1}	4.572×10^{-1}	5.974×10^{-1}	2.785×10^{-3}	6.466×10^{-3}	2.908×10^{-3}	6.480×10^{-3}
2D Complex Geom.	2.965×10^{-1}	9.354×10^{-1}	5.472×10^{-1}	9.585×10^{-1}	1.913×10^{-3}	9.317×10^{-3}	3.438×10^{-3}	9.852×10^{-3}
3D Poisson's	6.453×10^{-2}	1.459×10^{-1}	1.173×10^{-1}	1.415×10^{-1}	1.245×10^{-3}	6.529×10^{-3}	2.438×10^{-3}	5.815×10^{-3}
2D Klein-Gordon	7.548×10^{-1}	1.081×10^{0}	6.432×10^{-1}	1.081×10^{0}	2.791×10^{-3}	7.691×10^{-3}	1.012×10^{-3}	2.405×10^{-3}

Let us consider the following one-dimensional Poisson's equation

$$u_{xx}(x) = -(15\pi)^2 \cos(15\pi x), \qquad x \in \Omega, \tag{20}$$

$$u(x) = \cos(15\pi x), \qquad x \in \partial\Omega, \tag{21}$$

where $\Omega = \{x \,|\, 0 \leq x \leq 1\}$ and $\partial\Omega$ is its boundary. The exact solution to the above problem is a sinusoidal nonlinear function $u(x) = \cos(15\pi x)$. Considering a neural network solution for the above equation as $\hat{u}(x;\theta)$ parameterized with θ, we write the residual form of this one-dimensional Poisson's equation as follows:

$$\mathcal{D}(x,\hat{u}) := \hat{u}_{xx}(x;\theta) + (15\pi)^2 \cos(15\pi x) \qquad x \in \Omega, \tag{22}$$

$$\mathcal{B}(x,\hat{u}) := \hat{u}(x;\theta) - \cos(15\pi x), \qquad x \in \partial\Omega. \tag{23}$$

Next, we use the above residual form of this differential equation to construct an objective function as proposed earlier in Eq.(13a)

$$\mathcal{L}(\theta) = \boldsymbol{\mathcal{D}}^T\boldsymbol{\mathcal{D}} + \hat{\boldsymbol{\lambda}}^T\mathcal{Q}(\boldsymbol{\mathcal{B}}) + \frac{\mu}{2}\mathcal{Q}(\boldsymbol{\mathcal{B}})^T\mathcal{Q}(\boldsymbol{\mathcal{B}}), \tag{24}$$

where μ is the penalty parameter and $\boldsymbol{\mathcal{D}}$ is an N_Ω-vector $\{\mathcal{D}(x^{(1)},\hat{u}^{(1)}),\cdots,\mathcal{D}(x^{(N_\Omega)},\hat{u}^{(N_\Omega)})\}$ viewed as a column vector representing the outputs of N_Ω number of collocation points sampled from Ω calculated on the residual form of the differential equation as in Eq. (22). $\boldsymbol{\mathcal{B}}$ is a column vector $\{\mathcal{B}(x^{(1)},\hat{u}^{(1)}),\mathcal{B}(x^{(2)},\hat{u}^{(2)})\}$ representing the outputs of two boundary points sampled from $\partial\Omega$ on the residual form the boundary conditions as in Eq. (23).

$\hat{\boldsymbol{\lambda}} \in \mathbb{R}^2$ is a vector of multipliers for the boundary conditions and superscript T represents transposition. Finally, $\mathcal{Q}$ is a distance function defined in Eq. (1). In contrast to our constrained optimization with the ALM, the composite objective function adopted in PINNs (i.e. Eq. 5) yields the following loss function for the current example

$$L(\theta) = \frac{1}{N_\Omega}\boldsymbol{\mathcal{D}}^T\boldsymbol{\mathcal{D}} + \frac{1}{2}\boldsymbol{\mathcal{B}}^T\boldsymbol{\mathcal{B}}. \tag{25}$$

Having constructed the objective functions using the constrained-optimization method in the present work and the composite approach adopted in PINNs, we design three networks in such a way that they have the same number of neurons and hidden layers to allow a fair comparison. We use six weight layers with 50 neurons per layer in all three neural network models. More specifically, we have six weight layers in our conventional neural network model. Similarly, our second neural network model with the original residual layers has a weight layer in the front with two residual layers and an output weight layer, which makes a total of six weight layers. For our last neural network model with modified residual layers, we have a weight layer succeeded by four modified residual layers and an output weight layer that amounts to six weight layers as well. All the mapping functions $\mathcal{P}(x)$ in the modified residual layers are identity operators. Therefore, all three models have the same number of neurons and the same number weight layers and are end-to-end trainable. For this problem, the parameters of the network are initialized randomly with the Xavier initialization technique [23]. We use the same hyperparameters and train all the models under the same training settings with both objectives as in Eq. (24) and Eq. (25). Optimizer and learning rate scheduler is kept the same as mentioned in section 3.4 We set the limiting penalty parameter $\mu^\infty = 100$. As for the collocation points, we randomly generate 654 points from across our domain with uniform probability along with two boundary conditions. The results from all three neural network architectures trained with the PINN and PECANN approaches are juxtaposed in Fig. 2. We observe from these results that the PINN model with a composite objective function is visibly sensitive to the neural network choice and benefits the most from the adoption of modified residual layers, whereas the PECANN model with equality-constrained optimization is qualitatively less sensitive to the choice of the neural network architecture and performs very well for all three networks. However, the impact of the neural networks

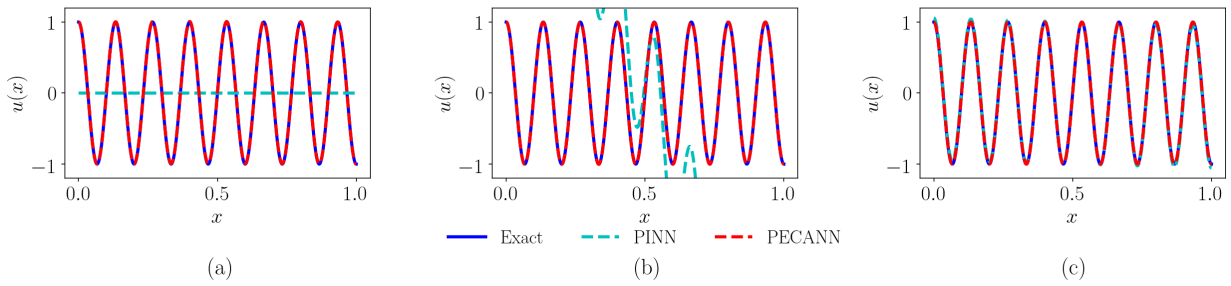

Figure 2: Performance comparison of PINN vs PECAN for different neural network architectures: (a) conventional neural network, (b) original residual neural network, (c) modified residual neural network. Note that PECANN approach converged with all network architectures while PINN only converged with our proposed modified residual neural network but with poor norms of errors

choice becomes much more clear by analyzing the error measures presented in Table 2, for both the PINN and PECANN models. Clearly, the use of the modified residual layers as part of the neural network architecture improves results for both PINN and PECANN models, but the best results, by and large, are obtained when the neural network architecture with modified residual layers is used as part of the PECANN model. Based on these results, we adopt the neural network architecture with modified residual layers (see Fig. 1d) for the rest of the problems reported in the present work.

In Fig. 3, we plot the loss values during training to gain a better understanding of improved results resulting from the use of modified residual layers in connection with the PECANN model. These results highlight that our PECANN model yields 7-10 orders of magnitude lower losses for the boundary conditions while efficiently producing lower losses on the PDE in comparison to the PINN approach. This is not entirely surprising, because

Table 2: Error measures resulting from the application of the PINN and PECANN models to three different neural network architectures considered in the present example. Errors are calculated separately for the solution domain Ω and the boundary $\partial\Omega$.

Architecture	PINN				PECANN			
	$L_{2,r} \in \Omega$	$L_\infty \in \Omega$	$L_{2,r} \in \partial\Omega$	$L_\infty \in \partial\Omega$	$L_{2,r} \in \Omega$	$L_\infty \in \Omega$	$L_{2,r} \in \partial\Omega$	$L_\infty \in \partial\Omega$
FC - NN	1.00×10^{0}	1.00×10^{0}	1.00×10^{0}	1.000×10^{0}	8.386×10^{-3}	8.867×10^{-3}	5.052×10^{-4}	6.580×10^{-4}
Orig. ResNET	4.775×10^{0}	5.838×10^{0}	5.838×10^{0}	5.838×10^{0}	3.381×10^{-3}	3.422×10^{-3}	2.390×10^{-4}	3.651×10^{-4}
Mod. ResNET	4.649×10^{-2}	5.741×10^{-2}	5.705×10^{-2}	5.741×10^{-2}	1.343×10^{-4}	1.643×10^{-4}	2.218×10^{-5}	3.136×10^{-5}

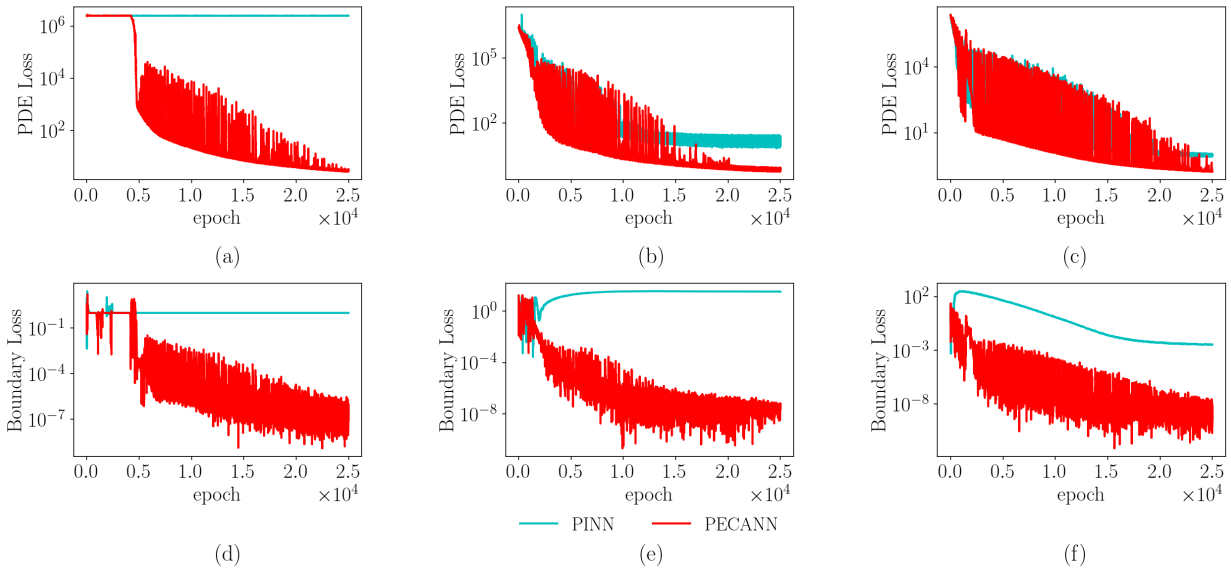

Figure 3: Training losses for PINN vs PECANN models. Top row: Physics (PDE) loss generated from adoption of different architectures: (a) conventional neural network, (b) original residual neural network, (c) modified residual neural network. Bottom row: Boundary loss generated from adoption of different architectures: (d) conventional neural network, (e) original residual neural network, (f) modified residual neural network.

the PECANN model is formulated to impose boundary conditions on the PDE as equality constraints in the optimization problem. This attribute of the PECANN model becomes evident when we compare the boundary and PDE losses collectively. We see from Fig. 3 that the PDE loss is not sacrificed when boundary losses are diminishing. Another important aspect of the PECANN model that we observe from the same plots is that the average loss rapidly reduces at the early stages of the training, which implies that the PECANN model can produce acceptable results with a small number of epochs.

To gain further insight into the intricacy of the PECANN model during training, we plot the evolution of learning rate decay, the multipliers, and the penalty parameter during training in Fig. 4(a)-(c), respectively. From the learning rate evolution (Fig. 4a), we observe that PECANN model has a higher learning rate in the second half of training epochs than PINN's learning rate. This implies that the optimizer has reached a fairly better minimum point than that of the PINN model. In contrast, the learning rate from the PINN model keeps decreasing during the entire training. This shows that the optimizer is still searching

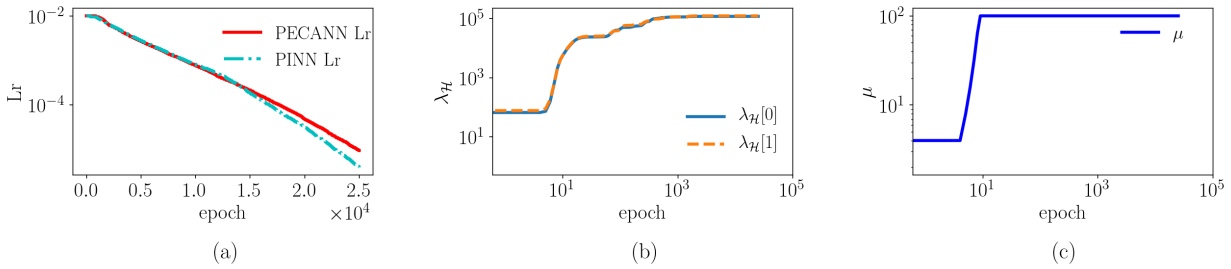

Figure 4: Hyperparameters during training for modified neural network architectures: (a) learning rates of the PINN and the PECANN models, (b) multipliers for the PECANN model, (c) penalty parameter for the PECANN model.Note the higher learning rate of the PECANN model in the second half of the training relative to the PINN model's learning rate.

for a better local minimum. From the penalty parameter curve in Fig. 4(c), we observe that the specified limiting penalty parameter is reached at an early stage during training. We further observe that while the penalty parameters reaches its limiting value, the multipliers still improves and plateaus much later during training as evident from Fig. 4(b). These observations substantiate the fact that both the multipliers and the penalty parameter induce convergence of ALM.

4.2. Two-dimensional Poisson's Equation

We consider a two-dimensional non-homogeneous Poisson's equation as follows:

$$\nabla^2 u(x,y) = -\frac{252\pi^2}{l^2}\cos(\frac{16\pi}{l}x)\exp(-\frac{2\pi}{l}y-\pi), \qquad (x,y)\in\Omega, \tag{26}$$

subject to the following boundary conditions

$$u(x,y) = \cos(\frac{16\pi x}{l})\exp(-\frac{2\pi}{l}y-\pi), \qquad (x,y)\in\partial\Omega \tag{27}$$

where $\Omega = \{(x,y) \mid -\frac{l}{2} \le x \le \frac{l}{2}, \frac{-l}{2} \le y \le \frac{l}{2}\}$ and $l = 2$ is the size of the domain. We manufacture a complex solution that oscillates along one side of the domain and dies out as we move away from the boundary. The exact solution is

$$u(x,y) = \cos(\frac{16\pi x}{l})\exp(-\frac{2\pi}{l}y-\pi), \qquad (x,y)\in\Omega. \tag{28}$$

The architecture and hyperparameters are kept the same as in section 3.4. The results obtained with the PINN and PECANN models are presented in Fig. 5. We observe that the

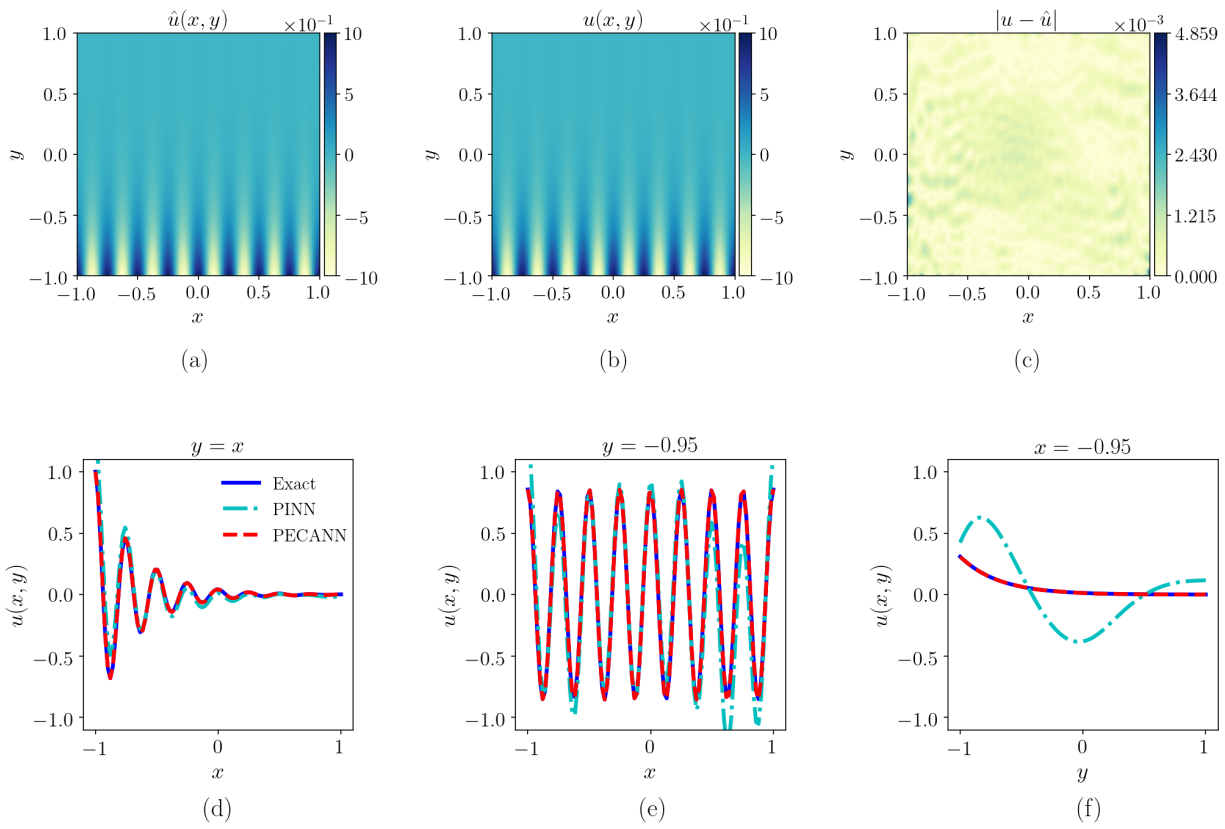

Figure 5: Two-dimensional Poisson's equation. Top row: Contour visualizations. (a) PECANN prediction, (b) exact solution, (c) absolute point-wise error for PECANN. Bottom row: Comparison of PINN and PECANN predictions against the exact solution along the lines (d) $y = x$, (e) $y = -0.95$, (f) $x = -0.95$.

point-wise absolute error distribution is uniform across the domain for the PECANN model. Since the PINN model has diverged, we do not portray its prediction for the entire domain. However, we compare our results with the PINN model by plotting the solution over lines horizontally, vertically, and diagonally, which demonstrates the divergence of the PINN model despite having a well-posed PDE with a unique solution and sufficiently expressive network.

4.3. Two-dimensional Helmholtz Equation

Let us now consider the following two-dimensional Helmholtz equation:

$$\nabla^2 u(x,y) + u(x,y) = (-\frac{200\pi^2}{l^2} + 1)\cos(\frac{2\pi}{l}x)\cos(\frac{14\pi}{l}y), \qquad (x,y) \in \Omega, \tag{29}$$

subject to the following boundary conditions

$$u(x,y) = \cos(\frac{2\pi}{l}x)\cos(\frac{14\pi}{l}y), \qquad (x,y) \in \partial\Omega, \tag{30}$$

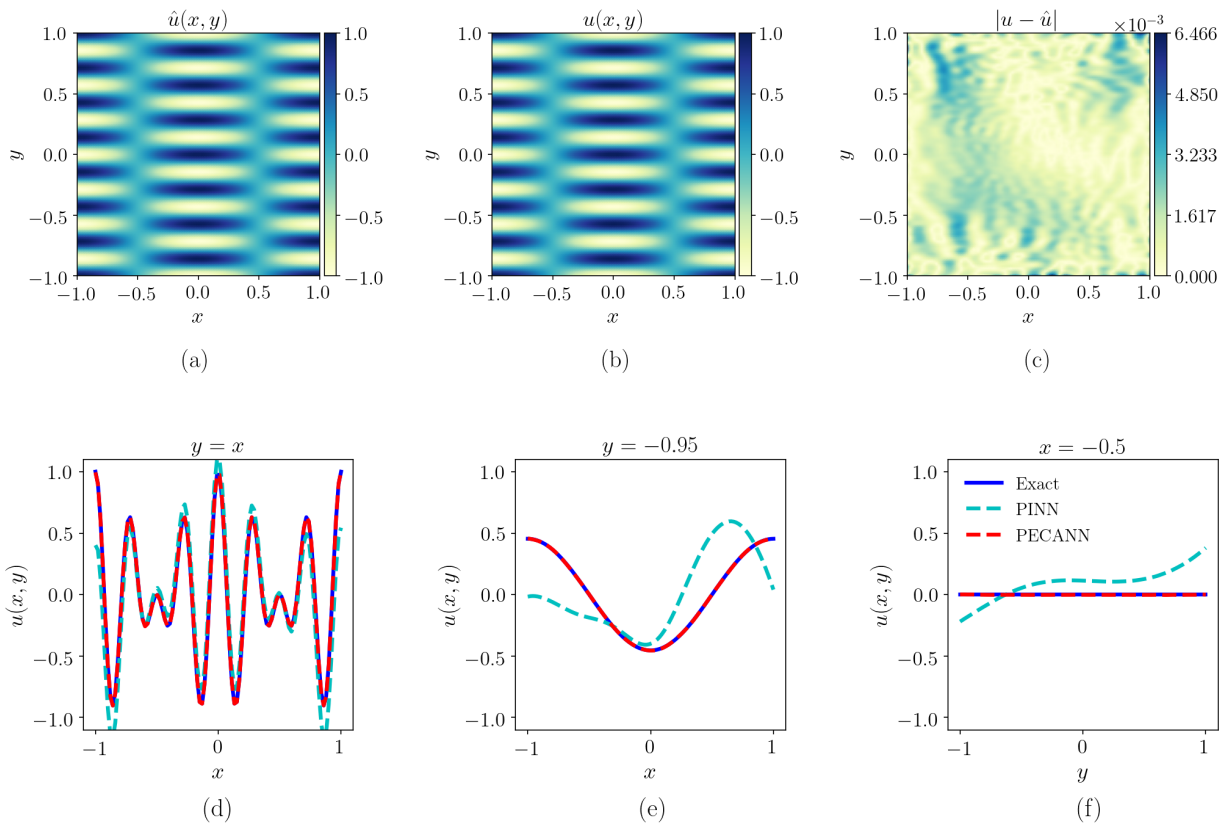

Figure 6: Two dimensional Helmholtz equation. Top row: contour visualizations. (a) PECANN prediction, (b) exact solution, (c) absolute point-wise error for PECANN. Bottom row: Comparison of PINN and PECANN predictions against the exact solution along lines (d) $y = x$, (e) $y = -0.95$, (f) $x = -0.50$.

where $\Omega = \{(x, y) \mid \frac{-l}{2} \leq x \leq \frac{l}{2}, \frac{-l}{2} \leq y \leq \frac{l}{2}\}$ and $\partial\Omega$ is its boundary and $u(x, y, t)$ is the unknown solution to be determined. Following the equation presented above, we manufacture an oscillatory solution that satisfy the Helmholtz equation Eq. (29) with its boundary conditions Eq. (30) as follows

$$u(x, y) = \cos(\frac{2\pi x}{l}) \cos(\frac{14\pi y}{l}), \qquad (x, y) \in \Omega. \tag{31}$$

As illustrated in Fig. 6(a), the PECANN model provides accurate predictions to the underlying solution with uniform error distribution across the domain as shown in Fig. 6(c). Since the PINN model fails to converge for this problem, the predictions for the entire domain are not displayed. However, to attest to the failure of PINN method, we portray plots over diagonal, horizontal and vertical lines in the domain in Fig. 6(d)(e)(f).

4.4. Complex Domain Example: Two-dimensional Steady-state Heat Conduction

Up to this point, we have tackled two-dimensional PDEs on square domains. However, neural network based methods rely on a series of collocation points, which avoids the onerous task of domain discretization for complex domains. They can be viewed as meshless methods to solve PDEs. To illustrate this aspect of neural network based methods, we solve the following two-dimensional steady-state heat equation on an irregular domain:

$$\nabla^2 u = -37\pi^2 \cos(\pi x)\cos(6\pi y), \qquad (x, y) \in \Omega, \tag{32}$$

subject to the following boundary conditions

$$u(x, y) = \cos(\pi x)\cos(6\pi y), \qquad (x, y) \in \partial\Omega, \tag{33}$$

where $\Omega = \{(x, y) \mid x = 0.55\rho(\theta)\cos(\theta), y = 0.75\rho(\theta)\sin(\theta)\}$ and $\rho(\theta) = 1 + \cos(\theta)\sin(4\theta)$ for $0 \leq \theta \leq 2\pi$. We manufacture a complex oscillatory solution for the governing equation Eq. (32) and its boundary conditions Eq. (33) as follows,

$$u(x, y) = \cos(\pi x)\cos(6\pi y), \qquad (x, y) \in \Omega. \tag{34}$$

For this problem, we keep the same hyperparameters and network architecture. The limiting penalty parameter μ^∞ is set as 1000 for this problem. For the present case, the predictions of both models for the entire domain are juxtaposed in Fig. 7.

As presented in Fig. 7(a), predicted solution from the PINN model is visibly different from the exact solution shown in Fig. 7(b). This difference is more pronounced on the boundaries as can be seen from the error distribution in Fig. 7(c). Fig. 7(d) presents the predictions from the PECANN model with much higher prediction accuracy than the PINN model due to constraining the boundary conditions in a principled fashion in the optimization problem. This key aspect is confirmed by the uniform error distribution of our method in the domain as shown in Fig. 7(f).

4.5. Nonlinear Inverse-PDE Problem

Our PECANN model is also suitable to solve inverse-PDE problems using multi-fidelity data. With multi-fidelity, we mean that the observed data may include both data with low

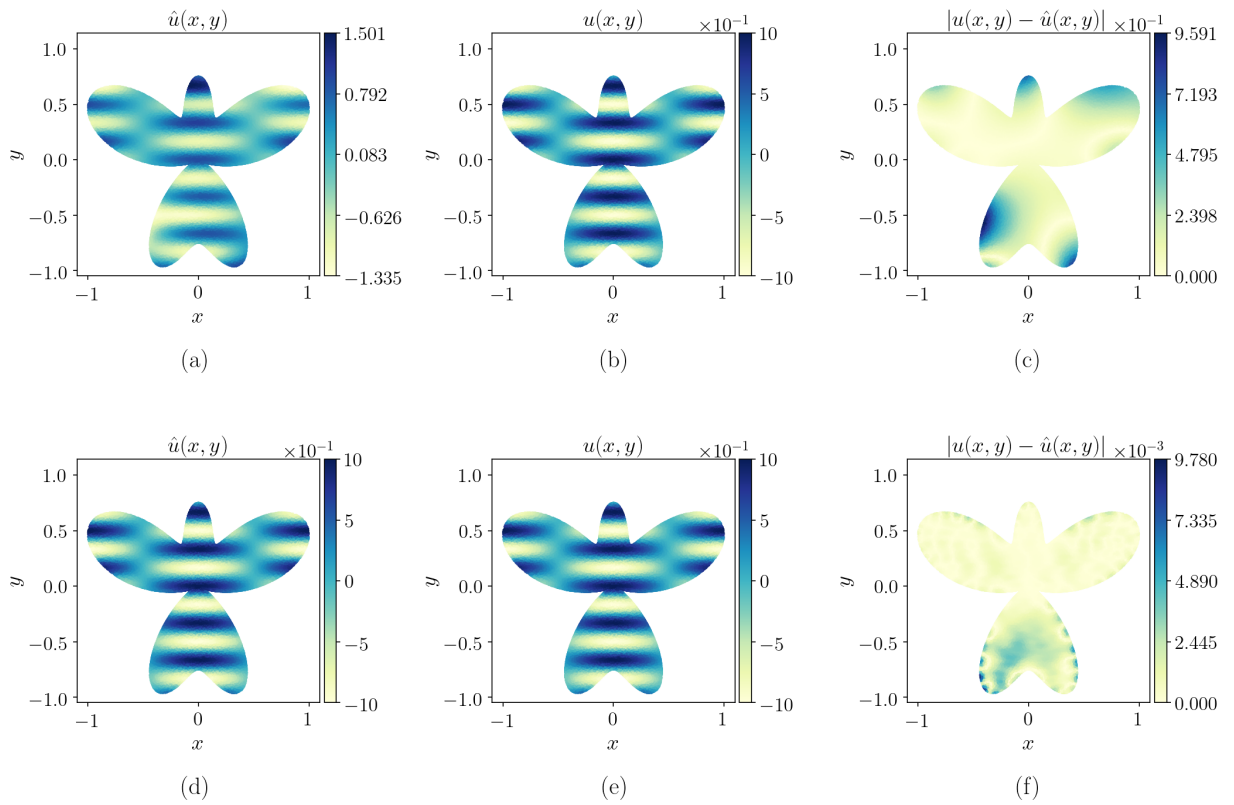

Figure 7: Two dimensional steady-state heat conduction problem with complex boundaries. (a) PINN prediction, (b) exact solution, (c) absolute point-wise error for PINN, (d) PECANN prediction, (e) exact solution, (f) absolute point-wise error for PECANN. Note the larger range in the colorbar in (a).

accuracy and data with very high accuracy. As part of our objective function formulation, we can constrain the high-fidelity data in a principled fashion and take advantage of the low-fidelity data to regularize our hypothesis space. To demonstrate our framework, we study one of the example problem that was tackled in Meng and Karniadakis [53] with composite neural networks. This particular inverse-PDE problem arises in unsaturated flows as they are central in characterizing contaminant transport [75], soil-atmosphere interaction [76], soil-plant-water interaction [77], ground-subsurface water interaction zone [78] to name a few. Describing processes involving soil-water interactions at a microscopic level is very complex due to the existence of tortuous, irregular, and interconnected pores [79]. Therefore, these flows are generally characterized in terms of their macroscopic characteristics. An important quantity that is essential in describing flows through unsaturated soil is hydraulic conductivity, which is a nonlinear parameter that is highly dependent on the geometry of the

porous media [79]. Let us consider the following nonlinear differential equation representing an unsaturated one-dimensional (1D) soil column with variable water content:

$$\frac{d}{dx}(K(h)\frac{dh(x)}{dx}) = 0, \qquad x \in \Omega, \tag{35}$$

subject to the following boundary conditions,

$$h(0) = -3, \tag{36a}$$

$$h(200) = -10, \tag{36b}$$

where $\Omega = \{x \mid 0 \leq x \leq 200 \text{ cm}\}$, $h(x)$ is the pressure head (cm) and $K(h)$ is the hydraulic conductivity (cm h^{-1}) which is described as follows:

$$K(h) = K_s S_e^{1/2}\Big[1 - (1 - S_e^{1/m})^m\Big]^2, \tag{37}$$

where K_s is the saturated hydraulic conductivity (cm h^{-1}), and S_e is the effective saturation expressed as follows [80]:

$$S_e = \frac{1}{(1 + |\alpha h|^n)^m}, m = 1 - 1/n, \tag{38}$$

where α is an empirical parameter that is inversely related to the air-entry pressure value (cm^{-1}) and m is an empirical parameter related to the pore-size distribution that is hard to measure due to the complex geometry of the porous media. We aim to infer the unknown empirical parameters α, and m from sparse measurements of pressure head h. To generate multi-fidelity synthetic measurements or experimental data, we select the soil type *loam* for which the empirical parameters are as follows: $\alpha = 0.036$ and $m = 0.36$.

We generate high-fidelity pressure data using the exact empirical parameters and low-fidelity data with $\alpha = 0.015$ and $m = 0.31$. Using the built-in `bvp5c` MATLAB function, we solve the governing PDE as given in Eq. Eqs. 35 through Eq. 38 using the selected empirical parameters to generate multi fidelity training data as shown in Fig 8(a). In Fig 8(b) we also depict the corresponding hydraulic conductivity $k(h)$ values for the pressure head data, which shows that low-fidelity hydraulic conductivity has a significant deviation from the exact hydraulic conductivity distribution. To highlight the robustness, efficiency, and accuracy of our framework on an inverse-PDE with multi-fidelity data fusion, we compare

our results with the results reported in Meng and Karniadakis [53]. For comparison purposes, we also choose a feed-forward neural network with two hidden layers with 20 neurons per layer as in [53] for their physics-informed neural network trained on high fidelity alone which failed to discover the parameters of interest. However, Meng and Karniadakis [53] constructed customized networks for high fidelity data and low fidelity data separately and then aggregated them together by manually crafted correlations. Therefore, they refer to their approach as a composite neural networks. Unlike Meng and Karniadakis [53], we do not need to make any inductive bias about the data and, therefore, use a single network initialized with Xavier initialization technique [23] that we separately train on high-fidelity and multi-fidelity data. This shows that robustness and efficiency of our approach that we can train the same network on multi-fidelity data without the need to design customized networks to process data differently. We let a single network discover and extract features from multi-fidelity data with the help of known physics. We use Adam with 10^{-2} initial learning rate and ReduceLROnPlateau(patience=500,factor=0.95) as our learning rate scheduler. We set the total number of epochs and the limiting penalty parameter μ^{∞} to 1500. As for the collocation points, we uniformly generate 400 collocation points from across our domain. As considered in [53], we assume the flux at the inlet q_0 is known, which allows us to use the integral form of Eq. (35) given as follows,

$$q(x) = -K(h)\frac{dh(x)}{dx} = q_0, \quad \frac{dq(x)}{dx} = 0. \tag{39}$$

Fig. 8(a) and Fig. 8(b) depict the reconstructed pressure head and the corresponding hydraulic conductivity distributions obtained by our PECANN model trained on high-fidelity and multi-fidelity data separately. Comparing with the exact solution, it is seen that the inferred results are highly accurate, which shows the robustness and efficiency of our method. Furthermore, in Table 3, we report the average and standard deviation of inferred α and m from our model along with the results from Meng and Karniadakis [53].

For this inverse problem, the average CPU training time was approximately 5.3s using a Dell OptiPlex 5050 desktop with an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz processor and 8 GB of RAM. It is seen that the same neural network architecture trained with our

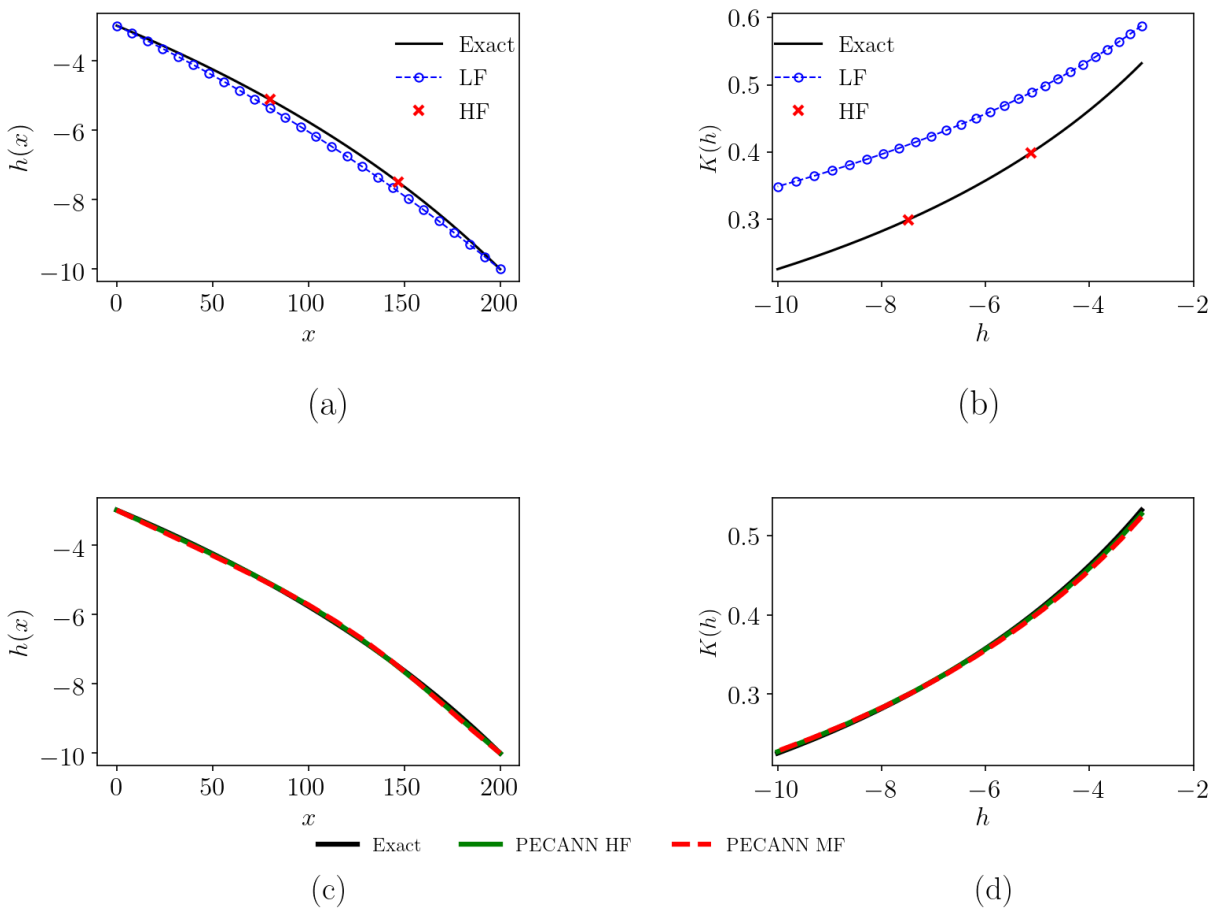

Figure 8: Parameter inference on multi-fidelity data for unsaturated flow through porous media: (a) low fidelity (LF) and high fidelity (HF) pressure head training data, (b) inferred hydraulic conductivity corresponding to low-fidelity and high-fidelity data, (c) pressure head reconstruction by PECANN model trained on high-fidelity and multi-fidelity data separately, (d) reconstructed hydraulic conductivity by PECANN model trained by high-fidelity and multi-fidelity data separately.

proposed approach on multi-fidelity data has improved averaged values for α and m which shows the importance of low-fidelity data. However, the same model trained on just a high-fidelity model has better more stable predictions since the data considered are high fidelity. However, owing to processing the low-fidelity data, our model trained on multi-fidelity data has slightly increased CPU time compared to the same model trained on just high fidelity data.

Table 3: Summary of the inferred parameters from using high-fidelity (HF) only or multi-fidelity (MF) data in the learning process averaged over ten different runs. Note that PECANN model training time is only 5 seconds on CPU averaged over 10 different runs

Models	Avg. α	$\sigma(\alpha)$	Avg. m	$\sigma(m)$	Avg. training time (s)
Ref. [53] with HF data only	0.0440	-	0.377	-	-
PECANN with HF data only	**0.0351**	0.0018	**0.3539**	0.0074	**5.3162**
Ref. [53] with MF data	0.0337	7.91×10^{-4}	0.349	0.0037	-
PECANN with MF data	**0.0361**	0.0086	**0.3574**	0.0332	**5.4943**
Exact value	0.0360	-	0.360	-	-

5. Conclusion

Physics-informed neural networks (PINNs) have been proposed to learn the solution of PDEs for forward and inverse problems. In the original PINN approach, the residual form of the PDE of interest and its boundary conditions are lumped into a composite objective function as mean-squared error as an unconstrained optimization problem. This objective function is then used to train a deep feed-forward neural network with an efficient optimizer of choice. Here, we have shown that objective function formulation is the root cause of the poor performance of the PINN approach when applied to learn the solution of more challenging multi-dimensional PDEs. To address this issue, we introduce physics- and equality-constrained artificial neural networks (PECANN), in which we pursue a constrained-optimization technique to formulate the objective function in the first place. Our approach is versatile for both forward and inverse problems and can fuse multi-fidelity data in the learning process in a principled fashion. Specifically, we adopt the augmented Lagrangian method (ALM) to constrain the PDE solution with boundary and initial conditions, and with high-fidelity data, if available. The objective function formulation in the PECANN model is sufficiently general to admit low-fidelity data to regularize hypothesis space in inverse problems as well.

Conventional feed-forward neural networks are known to suffer from the so-called vanishing gradient problem, which stalls the learning process. This issue can be very limiting for learning the solution of PDEs. To address this problem, we modified the original residual layers that

were proposed for image recognition tasks in a way that is unique to learning the solution of PDEs. Our modification with identity skip connections and a single weight layer and a *tanh* activation function can be viewed as a leaner version of the original residual layer and was found to be very effective in improving the accuracy of the PDE predictions for both the original PINN model and our PECANN model.

We have demonstrated the versatility and efficacy of our framework by solving several multi-dimensional PDEs, including a three-dimensional Poisson's equation and a time-dependent Klein-Gordon equation. We have also showcased the flexibility and robustness of our framework for learning from multi-fidelity data by considering a nonlinear inverse-PDE example. For all the problems considered, PECANN model produced results that are in excellent agreement with exact solutions, while the PINN approach failed to produce acceptable predictions.

Our findings suggest that not only the choice of the neural network architecture, but also the optimization problem formulation is crucial in learning PDEs using artificial neural networks. We conjecture that future progress in physics-constrained (informed) learning of PDEs would come from exploring new approaches in the field of non-convex constrained optimization field. Future endeavours could shed light on challenging questions such as: how does the loss landscape of neural networks change with respect to the optimization problem formulation? What is the optimal neural network architecture for PDE learning? And, is there a physics-based approach in searching for optimal architectures?

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. (1953204). Research was sponsored in part by the University of Pittsburgh, Center for Research Computing through the computing resources provided.

Appendix A. Additional Examples

Here, we present two additional PDE problems to further demonstrate the efficacy and versatility of our PECANN model.

Appendix A.1. Three-dimensional Poisson's Equation

We consider the following non-homogeneous three dimensional Poisson's equation in a cubic domain

$$\nabla^2 u(x,y,z) = -\frac{24\pi^2}{l^2}\cos(\frac{4\pi}{l}x)\cos(\frac{2\pi}{l}y)\sin(\frac{2\pi}{l}), \qquad (x,y,z)\in\Omega, \tag{A.1}$$

subject to the following boundary conditions

$$u(x,y,z) = \cos(\frac{4\pi}{l}x)\cos(\frac{2\pi}{l}y)\sin(\frac{2\pi}{l}z), \qquad (x,y,z)\in\partial\Omega, \tag{A.2}$$

where $\Omega = \{(x,y,z) \mid -l/2 \le x \le l/2, -l/2 \le y \le l/2, -l/2 \le z \le l/2\}$. The length of each side is set to $l = 2$. Similar to earlier example problems, we manufacture a solution that obeys the governing eq.(A.1) and the given boundary conditions (A.2) as follow,

$$u(x,y,z) = \cos(\frac{4\pi}{l}x)\cos(\frac{2\pi}{l}y)\sin(\frac{2\pi}{l}z), \qquad (x,y,z)\in\Omega, \tag{A.3}$$

We employ the same modified residual neural network as in the earlier problems, but with three input neurons to fit the input dimension. The hyperparameters are the same as discussed in section 3.4. We set the limiting value for the penalty parameter μ^∞ to 1000. The approximated solutions by both the baseline PINN and the proposed PECANN model are presented in Fig.A.9. Since PINN model did not converge, we only provide plots over a line drawn between two opposing corners in the domain as in Fig. A.9(c), and plots over lines connecting two opposing corners on plane sections as can be seen in Figs. A.9(d) and (f). From these line plots, we observe that large errors of the baseline PINN model arises particularly on the boundaries. Furthermore, we display relative $L_{2,r}$ errors and L_∞ errors of both models in the domain and on boundary in Table 1, which indicates that the predictions from our PECANN model is 2-3 orders of magnitude better than the results from the PINN model. The improvements are even more pronounced on the boundaries, underlining the success of our equality-constrained optimization formulation.

Appendix A.2. Two-dimensional Klein-Gordon Equation

We consider a nonlinear time-dependent problem, known as the Klein-Gordon equation, which is a hyperbolic PDE and is a key equation in many problems such as particle physics,

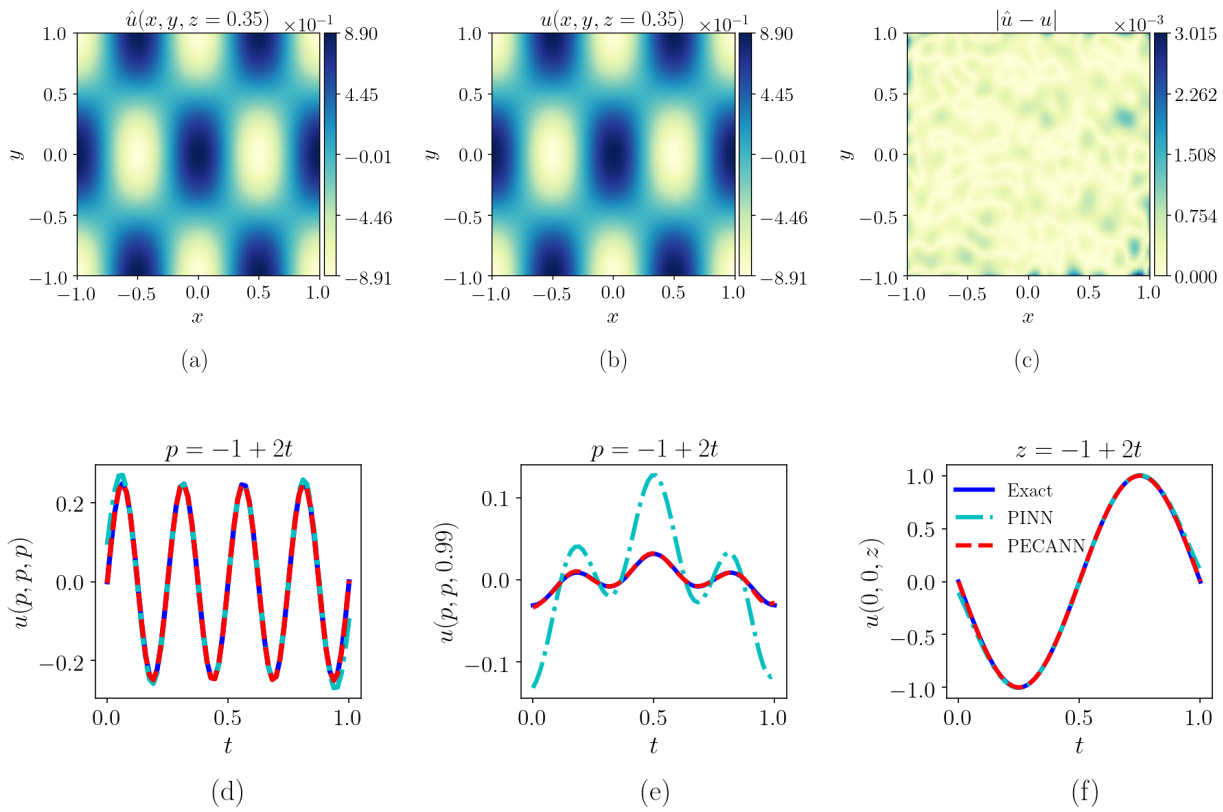

Figure A.9: Predictions for a three-dimensional Poisson's equation. Top row: Contour visualizations at $z = 0.35$ plane. (a) PECANN prediction, (b) exact solution, (c) absolute point-wise error for PECANN. Comparison of PINN and PECANN predictions against the exact solution over a line parameterized by (d) $x = -1 + 2t,\ y = -1 + 2t,\ z = -1 + 2t,\ \forall t = [0, 1]$, (e) $x = -1 + 2t,\ y = -1 + 2t,\ z = -0.5,\ \forall t = [0, 1]$, (f) $x = 0.5,\ y = 0,\ z = -1 + 2t,\ \forall t = [0, 1]$.

astrophysics, cosmology, classical mechanics. In this equation, we have different types of constraints that arise from physical boundary conditions and initial conditions. Here, we show that the PECANN model is applicable to equality constraints arising from both the boundary conditions and initial conditions.

$$u_{tt}(x,t) - u_{xx}(x,t) + u(x,y)^2 = F(x,t), \qquad (x,t) \in \Omega, \tag{A.4}$$

subject to the the following initial conditions

$$u(x,0) = \cos(2\pi x), \qquad 0 \leq x \leq 1, \tag{A.5a}$$

$$u_t(x,0) = 0, \qquad 0 \leq x \leq 1, \tag{A.5b}$$

and boundary conditions,

$$u(0,t) = \cos(2\pi t), \qquad 0 \leq t \leq 1, \tag{A.6a}$$

$$u(1,t) = \cos(2\pi t) + t^2, \qquad 0 \leq t \leq 1, \tag{A.6b}$$

where $\Omega = \{(x,t) \mid 0 \leq x1, 0 \leq t \leq 1\}$. For the purpose of demonstrating the effectiveness of our approach, we manufacture a solution that satisfy Klein-Gordon Eq. (A.4) and the boundary/initial conditions as in Eqs. (A.6b) and (A.5b) as follows :

$$u(x,t) = \cos(2\pi t)\cos(2\pi x) + xt^2, (x,t) \in \Omega. \tag{A.7}$$

we use the manufactured exact solution to generate the source function $F(x,t)$ in the governing eq.(A.4). All the hyperparameters are the same as mentioned in section 3.4. Fig.A.10(a) shows that the PECANN model is in agreement with the underlying exact solution as shown in Fig.A.10(b). As the PINN model fails to converge, we avoid presenting the predictions for the entire domain. However, we depict predictions from both models over a diagonal line defined with $t = x$ and a second line defined with $x = 0.5$ to represent the evolution of the predictions in time. From these plots, we observe that the PINN model diverges while our PECANN model does not show any deviation from the exact solution. We also portray a line plot across the spatial domain defined with $t = 0.90$ while keeping the time fixed. This result shows a similar behavior as before. In Table 1 we report $L_{2,r}$ and L_∞ errors in the domain and on the boundary, which indicates superior predictions of our PECANN model over our PINN model. We reiterate that this improved results are obtained thanks to constraining the boundary/initial conditions using Augmented Lagrange Method and our proposed modified residual neural network architecture.

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neur. In. 25 (2012) 1097–1105.

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

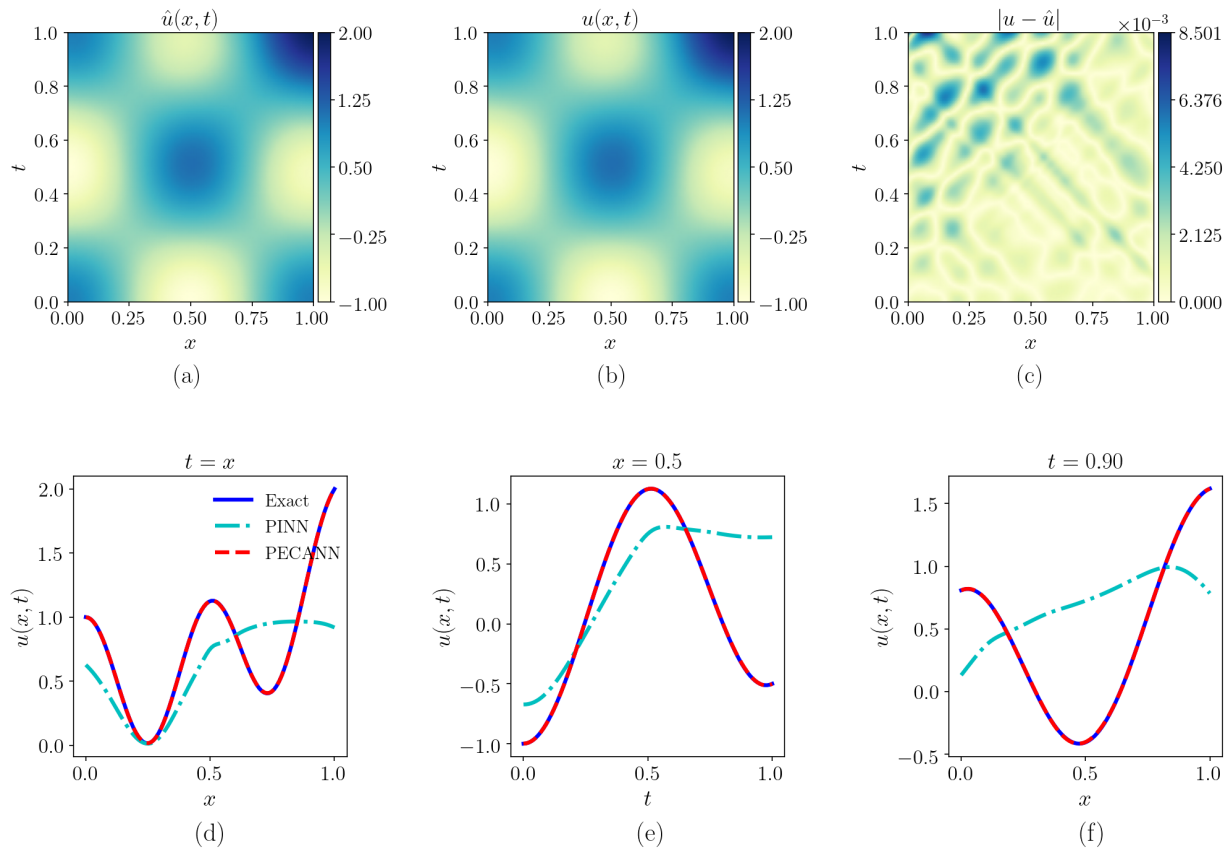

Figure A.10: *Klein-Gordon equation:* (a) predicted solution $\hat{u}(x, y)$ from PECANN, (b) exact solution $u(x, y)$, (c) absolute point-wise error, (d) comparison of exact vs. predicted solutions from PINN model and PECANN model over $t = x$ line, (e) comparison of exact vs. predicted solutions from PINN model and PECANN model over $x = 0.5$ line, (f) comparison of exact vs. predicted solutions from PINN model and PECANN model over $t = 0.90$ line.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal Process. Mag. 29 (2012) 82–97. doi:10.1109/MSP.2012.2205597.

[4] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, CoRR abs/1409.3215 (2014). arXiv:1409.3215.

[5] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473 (2014).

[6] J. Weston, S. Chopra, K. Adams, #TagSpace: Semantic embeddings from hashtags, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1822–1827. doi:10.3115/v1/D14-1194.

[7] H. Wang, B. Raj, On the origin of deep learning, arXiv preprint arXiv:1702.07800 (2017).

[8] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117. doi:10.1016/j.neunet.2014.09.003.

[9] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.

[10] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychol. Rev. 65 (1958) 386.

[11] D. H. Hubel, T. N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, J. Physiol. (Lond.) 160 (1962) 106.

[12] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biol. Cybern. 36 (1980) 193–202.

[13] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org.

[14] M. Jordan, Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986, Technical Report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.

[15] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[16] M. Schuster, K. K. Paliwal, Bidirectional recurrent neural networks, IEEE Trans. Signal Process 45 (1997) 2673–2681.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Commun. ACM 63 (2020) 139–144. doi:10.1145/3422622.

[18] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434 (2015).

[19] M. Mirza, S. Osindero, Conditional generative adversarial nets, arXiv preprint arXiv:1411.1784 (2014).

[20] P. Baldi, K. Hornik, Neural networks and principal component analysis: Learning from examples without local minima, Neural Netw. 2 (1989) 53–58.

[21] S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, Contractive auto-encoders: Explicit invariance during feature extraction, in: Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, Omnipress, Madison, WI, USA, 2011, p. 833–840.

[22] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004. doi:10.1017/CBO9780511804441.

[23] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Y. W. Teh, M. Titterington (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of *Proceedings of Machine Learning Research*, PMLR, Chia Laguna Resort, Sardinia, Italy, 2010, pp. 249–256.

[24] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: 2015 IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society, Los Alamitos, CA, USA, 2015, pp. 1026–1034. doi:10.1109/ICCV.2015.123.

[25] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: a survey, J Mach. Learn. Res. 18 (2018).

[26] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747 (2016).

[27] H. H. Tan, K. H. Lim, Review of second-order optimization techniques in artificial neural networks backpropagation, IOP Conference Series: Materials Science and Engineering 495 (2019) 012003. doi:10.1088/1757-899x/495/1/012003.

[28] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.

[29] M. Leshno, V. Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Netw. 6 (1993) 861–867.

[30] H. Lee, I. S. Kang, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1990) 110–131. doi:10.1016/0021-9991(90)90007-n.

[31] M. W. M. G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Meth. Eng. 10 (1994) 195–201.

[32] B. P. van Milligen, V. Tribaldos, J. A. Jiménez, Neural network differential equation and plasma equilibrium solver, Phys. Rev. Lett. 75 (1995) 3594–3597.

[33] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998) 987–1000.

[34] A. Meade, A. Fernandez, Solution of nonlinear ordinary differential equations by feedforward neural networks, Math Comput. Model. 20 (1994) 19–44. doi:10.1016/0895-7177(94)00160-X.

[35] C. Monterola, C. Saloma, Solving the nonlinear schrodinger equation with an unsupervised neural network, Opt. Express 9 (2001) 72–84.

[36] D. Parisi, M. C. Mariani, M. Laborde, Solving differential equations with unsupervised neural networks, Chem. Eng. Process. 42 (2003) 715–721.

[37] M. Quito Jr, C. Monterola, C. Saloma, Solving N-body problems with neural networks, Physical review letters 86 (2001) 4741.

[38] M. Hayati, B. Karami, Feedforward neural network for solving partial differential equations, J. Appl. Sci. 7 (2007) 2812–2817.

[39] W. E, B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12. doi:10.1007/s40304-018-0127-z.

[40] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. U.S.A 115 (2018) 8505–8510. doi:10.1073/pnas.1718942115.

[41] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.

[42] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81. doi:10.1016/j.jcp.2019.05.024.

[43] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[44] J. Nocedal, Updating quasi-Newton matrices with limited storage, Math. Comput. 35 (1980) 773–782.

[45] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A system for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, USENIX Association, USA, 2016, p. 265–283.

[46] M. Raissi, A. Yazdani, G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (2020) 1026–1030.

[47] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, J Mach. Learn. Res. 19 (2018) 932–955.

[48] M. Raissi, Z. Wang, M. S. Triantafyllou, G. E. Karniadakis, Deep learning of vortex-induced vibrations, J. Fluid Mech. 861 (2019) 119–137.

[49] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, P. Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Comput. Method. Appl. Mech. Eng. 358 (2020) 112623.

[50] A. Yazdani, L. Lu, M. Raissi, G. E. Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput. Biol. 16 (2020) e1007575.

[51] B. M. de Silva, J. Callaham, J. Jonker, N. Goebel, J. Klemisch, D. McDonald, N. Hicks, J. N. Kutz, S. L. Brunton, A. Y. Aravkin, Physics-informed machine learning for sensor fault detection with flight test data, arXiv preprint arXiv:2006.13380 (2020).

[52] Y. Liu, J. N. Kutz, S. L. Brunton, Hierarchical deep learning of multiscale differential equation time-steppers, arXiv preprint arXiv:2008.09768 (2020).

[53] X. Meng, G. E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems, Journal of Computational Physics 401 (2020) 109020.

[54] A. A. Ramabathiran, P. Ramachandran, SPINN: Sparse, physics-based, and partially interpretable neural networks for PDEs, J. Comput. Phys. 445 (2021) 110600. doi:10.1016/j.jcp.2021.110600.

[55] S. L. Brunton, B. R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508. doi:10.1146/annurev-fluid-010719-060214.

[56] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (2021) 422–440. doi:10.1038/s42254-021-00314-5.

[57] P. Grohs, F. Hornung, A. Jentzen, P. Von Wurstemberger, A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations, arXiv preprint arXiv:1809.02362 (2018).

[58] J. Darbon, G. P. Langlois, T. Meng, Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures, Res. Math. Sci. 7 (2020). doi:10.1007/s40687-020-00215-6.

[59] J. Berner, P. Grohs, A. Jentzen, Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations, SIAM J. Math. Data Sci. 2 (2020) 631–657.

[60] D. Z. Huang, K. Xu, C. Farhat, E. Darve, Learning constitutive relations from indirect observations using deep neural networks, J. Comput. Phys. 416 (2020) 109491. doi:10.1016/j.jcp.2020.109491.

[61] M. R. Hestenes, Multiplier and gradient methods, J. Optim. Theory Appl. 4 (1969) 303–320.

[62] M. J. Powell, A method for nonlinear constraints in minimization problems, in: R. Fletcher (Ed.), Optimization; Symposium of the Institute of Mathematics and Its Applications, University of Keele, England, 1968, Academic Press, London,New York, 1969, pp. 283–298.

[63] E. G. Birgin, J. M. Martínez, Practical Augmented Lagrangian Methods for Constrained Optimization, Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973365.

[64] R. van der Meer, C. W. Oosterlee, A. Borovykh, Optimally weighted loss functions for solving PDEs with neural networks, CoRR abs/2002.06269 (2020).

[65] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, arXiv preprint arXiv:2001.04536 (2020).

[66] I. Lagaris, A. Likas, D. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (2000) 1041–1049. doi:10.1109/72.870037.

[67] C. Monterola, C. Saloma, Characterizing the dynamics of constrained physical systems with an unsupervised neural network, Phys. Rev. E 57 (1998) R1247–R1250. doi:10.1103/physreve.57.r1247.

[68] D. P. Bertsekas, Multiplier methods: A survey, Automatica 12 (1976) 133–145.

[69] M. Bierlaire, Optimization : Principles and Algorithms, EPFL Press, CRC Press, Lausanne, Boca Raton, 2015.

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

[71] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S. G. Johnson, Physics-informed neural networks with hard constraints for inverse design, 2021. arXiv:2102.04626.

[72] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, Springer, 2016, pp. 630–645.

[73] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, 2019, pp. 8024–8035.

[74] P. J. Roache, Code verification by the method of manufactured solutions, J. Fluids Eng. 124 (2002) 4–10.

[75] A. A. Javadi, M. M. AL-Najjar, B. Evans, Flow and contaminant transport model for unsaturated soil, in: Theoretical and Numerical Unsaturated Soil Mechanics, Springer, 2007, pp. 135–141.

[76] N. An, S. Hemmati, Y. Cui, Numerical analysis of soil volumetric water content and temperature variations in an embankment due to soil-atmosphere interaction, Computers and Geotechnics 83 (2017) 40–51.

[77] V. Gadi, S. Singh, M. Singhariya, A. Garg, S. Sreedeep, K. Ravi, Modeling soil-plant-water interaction: effects of canopy and root parameters on soil suction and stability of green infrastructure, Engineering Computations (2018).

[78] M. Hayashi, D. O. Rosenberry, Effects of ground water exchange on the hydrology and ecology of surface water, Ground Water 40 (2002) 309–316.

[79] D. Hillel, Environmental soil physics: Fundamentals, applications, and environmental considerations, Elsevier, 1998.

[80] M. T. Van Genuchten, A closed-form equation for predicting the hydraulic conductivity of unsaturated soils, Soil Science Society of America Journal 44 (1980) 892–898.