INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 21, Number 5, Pages 652–673

doi: 10.4208/ijnam2024-1026

DEEP NEURAL NETWORK FOR SOLVING DIFFERENTIAL EQUATIONS MOTIVATED BY LEGENDRE-GALERKIN APPROXIMATION

BRYCE CHUDOMELKA, YOUNGJOON HONG*, JOHN MORGAN, HYUNWOO KIM*, AND JINYOUNG PARK

Abstract. In this paper, we propose the Legendre-Galerkin Network (LGNet), a novel machine learning-based numerical solver for parametric partial differential equations (PDEs) using spectral methods. Spectral methods leverage orthogonal function expansions, such as Fourier series and Legendre polynomials, to achieve highly accurate solutions with a reduced number of grid points. Our framework combines the advantages of spectral methods, including accuracy, efficiency, and generalization, with the capabilities of deep neural networks. By integrating deep neural networks into the spectral framework, our approach reduces computational costs that enable real-time predictions. The mathematical foundation of the LGNet solver is robust and reliable, incorporating a well-developed loss function derived from the weak formulation. This ensures precise approximation of solutions while maintaining consistency with boundary conditions. The proposed LGNet solver offers a compelling solution that harnesses the strengths of both spectral methods and deep neural networks, providing an effective tool for solving parametric PDEs.

Key words. Deep learning, neural network, spectral element method, Legendre-Galerkin method, data driven numerical method.

1. Introduction

Partial differential equations (PDEs) serve as fundamental tools for understanding natural phenomena. These mathematical equations describe diverse phenomena, ranging from fluid dynamics and electromagnetic fields to quantum mechanics and population dynamics. However, despite their significance, obtaining exact solutions for PDEs is often a formidable task. The complexity of PDEs necessitates the use of numerical methods and approximations. These techniques enable us to approximate solutions by discretizing the PDEs into algebraic systems of equations. However, traditional numerical methods often suffer from extensive computational costs, especially when generating a large number of numerical solutions as a database. In recent years, deep learning has emerged as a promising avenue for addressing the challenges associated with solving PDEs. By harnessing the power of artificial neural networks, deep learning techniques offer an alternative approach to tackle the computational inefficiencies encountered in traditional numerical methods. Through the integration of advanced machine learning algorithms and large-scale computational architectures, deep learning enables us to develop efficient numerical solvers for PDEs. In this study, we present a novel deep neural network (DNN) approach for solving differential equations using the Legendre-Galerkin approximation. Our methodology leverages accurate solutions in a supervised learning setting to find solutions, given a forcing function input, f. We employ the residual of the

Received by the editors on February 27, 2023 and, accepted on August 10, 2023.
2000 *Mathematics Subject Classification.* 65N35, 68T09, 68T07, 65Y10, 65N22.
*Corresponding authors.

weak formulation of the differential equations (DE) as a loss function. To demonstrate the effectiveness and versatility of our approach, we apply it to various types of equations, including fluid and wave models. By evaluating the numerical performance of the DNNs, we demonstrate their ability to provide accurate and efficient solutions.

Deep learning is a class of machine learning algorithms that employs multiple layers to progressively extract higher-level features from the dataset. These networks are referred to as *deep* due to their numerous hidden layers [2, 3, 11]. Neural networks (NNs) have demonstrated remarkable effectiveness in approximating continuous functions and achieving state-of-the-art performance in various fields [4, 9, 13, 17, 11, 38, 49]. Recent studies in computer vision have explored mathematical and numerical approaches of deep neural networks to enhance adversarial robustness [20, 42]. Our research is loosely connected to these investigations as we aim to gain a deeper understanding of implementing deep neural networks for numerical solutions. The central objective of our study is to examine the capability of a deep neural network in accurately approximating and predicting numerical solutions.

Previous works have demonstrated the success of neural architectures when applied to solutions of differential equations [4, 19, 30, 1, 49, 44, 46, 33, 36, 7]. There are some popular neural-network-based methods for solving high dimensional partial differential equations such as a Deep Ritz Method (DRM) [44, 48, 34] and the Deep Galerkin Method (DRM) [36]. The authors in [4, 49] introduce data-driven discretization, a method for learning optimized approximations to PDEs based on traditional finite-volume (or difference) schemes. The algorithm uses neural networks to estimate spatial derivatives, which are optimized end to end to best satisfy the equations on a low-resolution grid. Recently, the Physics Informed Neural Networks (PINN) introduces a novel methodology for finding solutions to complex dynamical systems utilizing automatic differentiation; see e.g. [30, 28, 19, 29, 26] among many other references. In [19, 47], the authors utilize the variational form, *i.e.*, the weak form to enhance the accuracy of their network. However, PINNs are designed to predict a single instance for a given set of PDEs. Consequently, when the input instance changes, the NNs need to be retrained in order to adapt to the new input data. More recently, an alternative approach is introduced learning the solution operator of a family of PDEs, which is defined by the map from the inputCinitial conditions and boundary conditions, to the output solution functions [25, 24, 43]. Our research differs from the previous work by exploring different neural network architectures to obtain an accurate solution to the parametric PDEs. We use a deep convolutional neural network (CNN) to achieve sufficiently accurate solutions by predicting coefficients of spectral approximation based on Legendre-Galerkin methods [12, 35]. This approach is partially related to the data-driven discretization in [4, 49], but their neural networks predict coefficients of numerical derivatives such as finite difference methods. In addition, they make use of standard finite volume schemes to compute numerical fluxes. Hence, the accuracy of numerical solutions generated by the finite volume method is limited even if high resolution methods are implemented.

While both our research and the DGM share the term "Galerkin", the methodology and results differ significantly. Unlike the DGM, which solely relies on taking integrals of the differential equations without multiplying test functions, our

approach aligns more closely with the conventional variational formulation, specifically the weak formulation. In contrast to the DGM, which predicts only a single instance, the LGNet represents a version of operator learning capable of predicting solutions for parametric PDEs. In addition, the DGM utilizes Monte Carlo sampling to handle high-dimensional problems during integration and differentiation. In contrast, the LGNet employs numerical differentiation and integration based on well-known quadrature rules. This choice enhances the accuracy of the LGNet compared to the DGM. When fitting the proposed LGNet, we introduce a residual of numerical integration inherited from a weak formulation of the differential equations. This choice is natural as the architecture mainly relies on spectral element methods constructed by the weak formulation of the differential equations. In addition, performing integration by parts reduces the order of derivatives in the differential equations. Hence, one can effectively avoid numerical errors introduced by numerical differentiation. Most importantly, the predicted solution obtained by the LGNet satisfies the exact boundary condition such as Dirichlet and Neumann boundary conditions. In most scientific machine learning frameworks, enforcing specific boundary conditions is often challenging, leading to inaccurate solutions. However, within the LGNet framework, the situation is different. Since each basis is created through a linear combination of Legendre polynomials, it inherently satisfies the exact boundary condition.

A network trained on noisy or corrupt data may suffer in accuracy, especially when applied to neural networks for PDEs. The SEM can achieve spectral accuracy with a relatively small number of collocation points. Hence, our data set will be as accurate as possible in order for the network to predict the dynamics. It is noteworthy that the methodology under consideration can incorporate various choices of polynomial basis functions into the novel neural network architecture. More precisely, there are many feasible choices of basis functions, such as Fourier series, Chebyshev polynomials, or Jacobi polynomials. Hence, our architecture is flexible and extendable to other numerical approximation. In addition, since the coefficients of the spectral approximation are predicted by the network, the main structure of the numerical approximation with the polynomial basis is maintained. In this regard, various existing numerical methods such as enriched space methods are applicable to the proposed network architecture by adding a proper basis function; for more details, see e.g. [14, 6].

In the calculations presented here, the model consists of convolutional layers connected with a nonlinear activation function between each layer. CNNs are widely used in various tasks, including facial recognition, object detection, semantic segmentation, natural language processing, and more recently, time-series data, to achieve state-of-the-art results [8, 21, 22, 27, 39]. CNNs involve a series of linear operations, which, in the context of a DNN, can be combined with nonlinear operations, making them effective universal function approximators [9, 17]. We train multiple configurations of CNN neural architectures with different nonlinear operations, referred to as activation functions, on accurate data sets generated with the SEM for supervised learning. Each configuration is then tested using an out-of-sample set of 1,000 randomly sampled solutions that are not present in the input data set. This allows us to measure the network's ability to generalize to unknown data. In this paper, we develop a version of CNNs incorporated with the Legendre-Galerkin framework to find a numerical solution of DEs using a deep

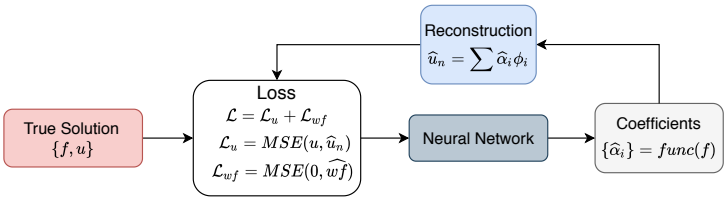


FIGURE 1. The Legendre-Galerkin Deep Neural Network Algorithm. This diagram demonstrates the training process for the LGNet. The network's input, the forcing function f, and the output, a set of coefficients $\{\widehat{\alpha}_i\}$, that are used to reconstruct the predicted solution, $\widehat{u}$. The next step in the training loop is to measure the loss, $\mathcal{L}$, between the ground truth solution u and the predicted solution. The weak form loss is computed by minimizing the weak form for the differential equation. The iterative process then repeats until finished.

neural network. The networks predict coefficients, α_i, of a spectral approximation consisting of Legendre polynomial basis functions φ_i, such that $u \simeq \sum_{i=0}^{N} \alpha_i \varphi_i$, where u represents a solution of the DE or PDE. Additionally, for accurate and reliable training, we utilize the residual of the weak formulation as a loss function. We implement different types of equations, including two-dimensional models, to demonstrate the effectiveness of our approach.

The article is organized as follows: In Section 2 we address the nonlinear function approximation of DNNs, and introduce a novel architecture of the DNNs based on numerical approximation. In Sections 3 and 4, linear and nonlinear models equipped with different boundary conditions are implemented including two-dimensional problems. A sequence of numerical experiments are presented to demonstrate the numerical performance of the proposed method. We conclude the paper with a summary section 5.

2. Data-Driven Numerical Methods

Consider a set of differential equations:

$$\begin{aligned} &\mathcal{F}(u, u_x, u_{xx}) = f(x), \quad x \in [-1, 1], \\ &\mathcal{B}(u, u_x) = g(x), \qquad \text{at} \quad x = -1, 1, \end{aligned} \tag{1}$$

where $\mathcal{F}$ is a linear or nonlinear operator and $\mathcal{B}$ is a boundary operator. Our objective is to construct a neural network that can learn a suitably accurate solution to the given differential equation by considering the forcing function.

One of the key benefits of the LGSEM is its ability to achieve spectral accuracy using a minimal number of nodal points [35, 12]. In other words, even with a relatively small set of polynomial basis functions, specifically a Legendre basis in this paper, the numerical errors are at the level of machine precision ($10^{-14} \sim 10^{-16}$). Given that the proposed framework relies on the LGSEM, we can obtain an accurate solution to (1) when compared to other scientific machine learning models.

Spectral methods depend on a global discretization approach. The conventional approach to implementing them involves approximating the numerical solution as a sum of global basis functions:

$$u(x) \simeq \sum_{k=0}^{N-1} \alpha_k \phi_k(x), \tag{2}$$

where N is the number of the global basis function. There exist numerous viable options for basis functions, including Fourier series, Chebyshev polynomials, and Legendre polynomials. In this paper, we specifically utilize the Legendre polynomials. One advantage of utilizing Legendre polynomials is their mutual orthogonality in the standard L^2 inner product, which simplifies computations. Consequently, the discrete variational formulations employing Legendre polynomials result in sparse matrices. Therefore, when deriving the loss function based on the weak formulation, our implementation becomes efficient. Furthermore, it is important to highlight that the LGSEM employs a linear combination of Legendre polynomial basis functions, which satisfy exact boundary conditions. By utilizing the same basis functions in our model, the numerical solution predicted by our model naturally conforms to exact boundary conditions, encompassing Dirichlet, Neumann, and Robin boundary conditions.

Let us define the Legendre polynomial, $L_k(x)$, on the Gauss-Lobatto quadrature, which plays an important role on numerical differentiation and integration. We set the global basis functions

$$\phi_k(x) := L_k(x) + a_k L_{k+1} + b_k L_{k+2}(x), \quad x \in [-1, 1], \tag{3}$$

where a_k and b_k are determined by the boundary conditions of the underlying differential equations. Applying the SEM, we can obtain an accurate numerical solution given by (2); for more details on the LGSEM, see e.g. [35]. The set of numerical solutions generated by the SEM is used for the training data set in our neural network.

For any given f, we propose a Legendre-Galerkin Deep Neural Network (LGNet) algorithm to find the coefficients, $\{\widehat{\alpha}_k\}$, from which we can reconstruct the corresponding predicted numerical solution, $\widehat{u}$, to the known numerical solution $u(x)$. If our random input data is given by

$$f(x) = m_1 \sin(\pi w_1 x) + m_2 \cos(\pi w_2 x), \tag{4}$$

where parameters, m_1, m_2, w_1, w_2, follow normal or uniform distributions, then the corresponding output will be

$$\{\widehat{\alpha}_k\} \implies \widehat{u} = \sum_{k=0}^{N-1} \widehat{\alpha}_k \phi_k, \tag{5}$$

where $\{\widehat{\alpha}_i\}$ describes the set of learned coefficients that are used to reconstruct the predicted numerical solution, $\widehat{u}_N$. A detailed description is provided in Figure 1, illustrating the network's input (the forcing function f) and output (a set of coefficients $\widehat{\alpha}_i$) utilized to reconstruct the predicted solution $\widehat{u}$ as shown in equation (5). The subsequent step in the training loop involves measuring the solution loss, denoted as $\mathcal{L}_u$, which directly uses the ground truth solution u and the predicted solution. The ground truth solution u is obtained using a standard spectral element method. Additionally, the weak form loss $\mathcal{L}_{wf}$ is computed by minimizing the

residual of the weak form for the given differential equation. This iterative process continues until completion.

We can generate data sets with an arbitrary number of solutions using the SEM. These solutions are then fed through the LGNet algorithm, reconstructed, and then the difference between the predicted solution and the actual solution are measured as the primary metric of performance. This can be done using either the mean absolute error defined as,

$$MAE\{h,\widehat{h}\} := \frac{1}{M}\sum_{i=0}^{M}\left|h_i-\widehat{h}_i\right|, \tag{6}$$

or using the mean squared error defined as,

$$MSE\{h,\widehat{h}\} := \frac{1}{M}\sum_{i=0}^{M}\left(h_i-\widehat{h}_i\right)^2, \tag{7}$$

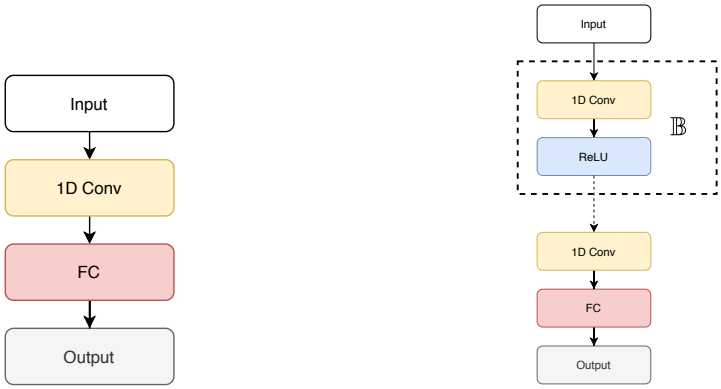
for M arbitrary functions, and its predicted function $\widehat{h}_i$. Other loss metrics can be used, such as the root mean square error or relative ℓ^2 error, but we found (6) and (7) to outperform others. It was observed that the mean square error provides a better metric for generalization of a neural network with regards to the accuracy of the predicted solution. The predicted coefficients for the global basis vectors, $\{\widehat{\alpha}_i\}$, were more accurate for (7) than (6). We found this to be a consequence of the sensitivity of reconstructing accurate solutions to the learned coefficients when using the SEM. Although the coefficients are not used in the loss metric, we did have access to them for reference indicating that the choice of basis is arbitrary and can be generalized to improve scalability.

Remark 2.1. *Other polynomials such as Chebyshev or Jacobi polynomials can be combined with the proposed DNN architecture since Chebyshev spectral Galerkin methods or Jacobi spectral Galerkin method have been well developed by those polynomial basis functions.*

We now construct the deep neural network based on convolutional neural networks. In [4], the authors introduced a CNN based on finite volume discretization. However, their method is limited to periodic boundary conditions and the accuracy is expected to be similar to or smaller than their training sets, which are generated using 2^{nd} or 3^{rd} order numerical methods. In contrast, our algorithm is easily extendable to various boundary conditions, including Dirichlet, Neumann, and Robin boundary conditions, as long as we choose the appropriate set of Legendre polynomials. It is important to note that the choice of a_k and b_k for the Legendre basis function in (3) ensures an exact boundary condition. After predicting the coefficients α_k, the numerical solution obtained from reconstruction satisfies the exact boundary condition. As a result, numerical errors arising from the boundary conditions become negligible.

3. Paradigm problem: one-dimensional linear models

We aim to observe the effectiveness of using a neural network (NN) to generalize and find solutions to differential equations. To validate the efficacy of this approach, it is appropriate to start with an example that is non-trivial yet not overly difficult. We refer to this example as our paradigm problem, as its equation



(A) `Linear` Architecture (B) `NetA` Architecture

FIGURE 2. The `NetA` neural network architecture is comprised of multiple blocks, $\mathbb{B}$, that add depth to the network. This diagram depicts a `NetA` architecture with just 1 block that is comprised of a 1D-convolutional operation with $\mathbb{F}$ filters followed by a `ReLU` operation. The final convolution in the network does not have the `ReLU` operation and is then flattened into a fully connected network. The output of this network is a set of coefficients, $\{\widehat{\alpha}_i\}$ that are then used to reconstruct the solution.

structure can be generalized to many other interesting problems, such as Burgers' equations (see Section 4). All of our experiments were conducted on an Intel Core i9-10900K processor with an NVIDIA GeForce RTX 2080 SUPER GPU.

3.1. Convection Diffusion Equation (CDE). To deliver our idea, we start with a paradigm problem:

$$\begin{cases} -\varepsilon u_{xx} - u_x = f, \\ u(-1) = u(1) = 0, \end{cases} \tag{8}$$

where $\varepsilon = O(1)$ is a diffusion parameter. The paradigm problem poses an interesting challenge for a neural network. Once the paradigm problem is solved, our algorithm can naturally be generalized to nonlinear, time-dependent, or 2D problems. Notice that homogeneous Dirichlet boundary conditions are used but, as we will come to see, the choice of boundary conditions do not have an effect on performance.

We look for basis functions as a compact combination of Legendre polynomials,

$$\phi_k(x) = L_k(x) + a_k L_{k+1} + b_k L_{k+2}(x), \tag{9}$$

where L_k is the k-th Legendre polynomial and the parameters a_k, b_k are chosen to satisfy the boundary conditions of the differential equations; see e.g. [35] for more details. Such basis functions are referred to as modal basis functions. Hence, for the homogeneous Dirichlet boundary conditions we have $a_k = 0$ and $b_k = -1$ in (3)

which yields

$$\phi_k(x) = L_k(x) - L_{k+2}(x). \tag{10}$$

The set of global basis functions, $\{\phi_j\}$, are then used as test functions in the weak formulation of the derivative. Noting that we have the zero boundary conditions at $x = \pm 1$, we multiply the DE in (8) by ϕ_j, and integrate from -1 to 1:

$$-\varepsilon \int_{-1}^{1} u_{xx}\phi_j - \int_{-1}^{1} u_x \phi_j \simeq \int_{-1}^{1} f\phi_j. \tag{11}$$

By integration by parts, we find that

$$\varepsilon \int_{-1}^{1} u_x (\phi_j)_x - \int_{-1}^{1} u_x \phi_j \simeq \int_{-1}^{1} f\phi_j. \tag{12}$$

In this way, we can avoid the 2^{nd} order derivative. Hence, we set the residual

$$\begin{gathered} LHS := \sum_{j=0}^{m} \left(\varepsilon \int_{-1}^{1} \widehat{u}_x (\phi_j)_x - \int_{-1}^{1} \widehat{u}_x \phi_j \right), \qquad RHS := \sum_{j=0}^{m} \left(\int_{-1}^{1} f\phi_j \right), \\ \mathcal{J}_{wf} := LHS - RHS, \end{gathered} \tag{13}$$

where $\widehat{u}$ is a numerical approximation of u, then minimize $\mathcal{J}_{wf}$ together with the other errors. Here, m represents the number of test functions in the weak formulation, which are utilized in the construction of the loss function. This loss function serves as a regularization factor [5, 40]. For numerical derivatives and integration, we employ spectral differentiations and Gauss-type integration formulas, which offer an exponential convergence rate. The Gauss-type quadrature formulas are powerful tools not only for evaluating integrals but also for performing spectral differentiations [32, 10]. Additionally, by precomputing the first-order differentiation matrix, the differentiation in physical space can be efficiently performed through matrix-matrix and matrix-vector multiplications. A similar approach was previously introduced by the authors in [30]; however, they solely employed the weak form for the loss function with automatic differentiation. In our method, we predict the coefficients of the spectral approximation. During the generation of the training dataset, these coefficients can be obtained from Galerkin methods that rely on the weak formulation.

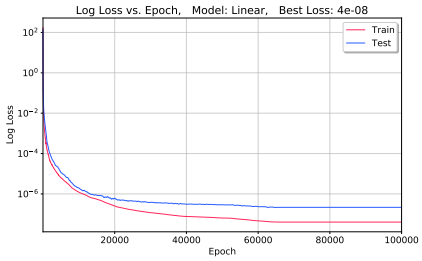
We measure the accuracy of our predicted solution, $\widehat{u}$, against the ground truth solution, u, using a loss function, $\mathcal{L}$, defined by

$$\mathcal{L} := \mathcal{L}_u + \mathcal{L}_{wf}, \tag{14}$$

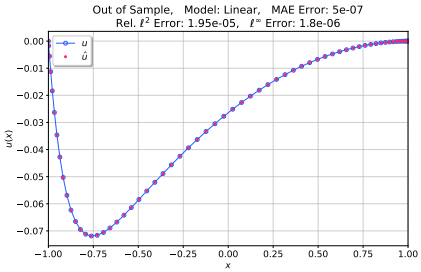
where the loss corresponding to the numerical solution and weak form are $\mathcal{L}_u$ and $\mathcal{L}_{wf}$, respectively, and defined by

$$\begin{aligned} \mathcal{L}_u &:= MSE\left\{u, \widehat{u}\right\}, \\ \mathcal{L}_{wf} &:= MSE\{LHS, RHS\}, \end{aligned} \tag{15}$$

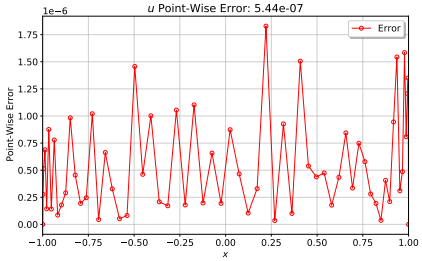
(A) Individual Losses



(B) Training Loss



(C) Predicted Solution $\widehat{u}$



(D) Point-wise Error

FIGURE 3. We trained a linear model (8) with $\varepsilon = 10^{-1}$ using a dataset with 10,000 solutions and 32 collocation points. The individual losses for $\mathcal{L}_u$ and $\mathcal{L}_{wf}$ are plotted on a semi-log plot over 100,000 epochs. The overall losses for the training and test sets can be seen on a semi-log plot beginning to separate as the number of epochs increases. An example of a predicted solution on out-of-sample data is observed with a mean relative ℓ^2 error on the order of 10^{-5}. The point-wise error plot shows a mean absolute error on the order of 10^{-7}.

3.1.1. Experimental Results. The paradigm equations, (8), can be solved with a linear model. The NN architecture can be seen in Figure 2a. Our architecture can achieve an average relative ℓ^2 error on the order of 10^{-5} on out-of-sample solutions with an extremely small dataset. Refer to Figure 3 for reference. For instance, by using a training dataset with 10,000 randomly sampled solutions, our architecture effectively predicts sufficiently accurate solutions to Equation (8).

For a linear model with 32 convolutional filters with a kernel size of 5, we were able to obtain an average relative ℓ^2 error over an out-of-sample data set with 1,000 solutions on the order of 10^{-6}. An example of an out-of-sample predicted solution can be seen in Figure 3c. Our model generalizes well to out-of-sample data and is reflected in a log-loss vs. epoch training loss plot in Figure 3b. The individual losses were tracked in Figure 3a as a function of epoch and the loss associated with $\widehat{u}$ is a couple of orders of magnitude greater than the loss associated with the weak form. The out-of-sample generalization is improved by increasing the size of input training set but the improvements are marginal at best.

3.1.2. Discussion. When a sufficient number of training data sets is provided (e.g., $1,000$), the best performing models were on the order of 10^{-5} or 10^{-6}. While varying the architectural parameters might lead to marginal performance improvements, theoretical considerations should also be addressed. Introducing a pooling operation could enhance overall accuracy, but it has not been implemented here. Increasing the size of the training set could also improve performance; however, there is a trade-off due to GPU memory limitations for training neural networks, which might necessitate switching from L-BFGS to a stochastic gradient method.

To offer contrast, we did compare the performance of linear models to nonlinear models. Linear models converge rather quickly due to the L-BFGS optimization method but nonlinear models did not. Perhaps given enough time the models would converge but linear models outperformed nonlinear models, *i.e.*, models with nonlinear activation functions do not perform well on linear DEs. All nonlinear models that we trained and evaluated drastically under-performed linear models [23].

3.2. Helmholtz Equation. Next, we will utilize our algorithm to find solutions to the Helmholtz equation with Neumann boundary conditions. This equation exhibits wave phenomena, as it is a form of the wave equation with significant applications in the field of optics [37]. Moreover, the utilization of different boundary conditions expands the notion of the general applicability of our algorithm. Consider the Helmholtz differential equation given as

$$\begin{cases} u_{xx} + k_u u = f(x), \\ u'(-1) = u'(1) = 0, \end{cases} \tag{16}$$

where f is defined as in (4). We remark that the governing equation, (16), describes essential aspects of the scattering of linear waves by periodic multiply layered gratings. In fact, by considering the quasiperiodicity of solutions, the transverse electric and magnetic waves can be reduced to one-dimensional inhomogeneous Helmholtz equations through generalized Fourier (Floquet) series expansions. These equations can be solved using numerical approaches, as demonstrated in works such as [15, 16].

The global basis function, which is a linear combination of Legendre polynomials, satisfies the homogeneous Neumann boundary conditions. For this, in (9), we set

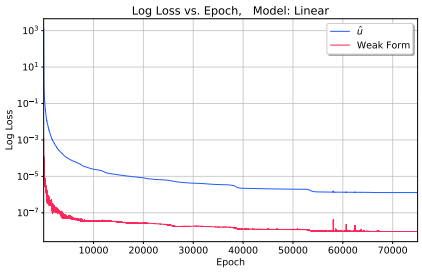
$$a_k = 0, \quad \text{and} \quad b_k = -\frac{k(k+1)}{(k+2)(k+3)}$$

in (3) which yields

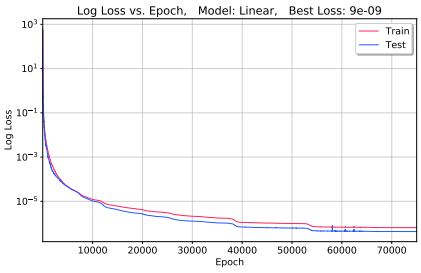
$$\phi_k(x) = L_k(x) - \frac{k(k+1)}{(k+2)(k+3)} L_{k+2}(x), \tag{17}$$

where $0 \leq k \leq N-2$; for more details, see e.g. [35]. The set of global basis functions, $\{\phi_k\}$, are then used as test functions in the weak formulation,

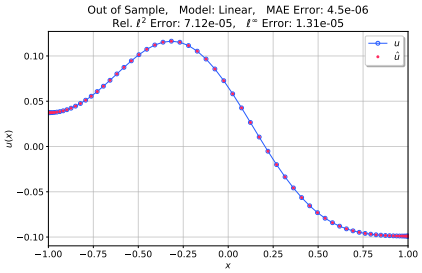
$$-\int_{-1}^{1} u_x (\phi_k)_x dx + k_u \int_{-1}^{1} u \phi_k dx = \int_{-1}^{1} f \phi_k dx. \tag{18}$$



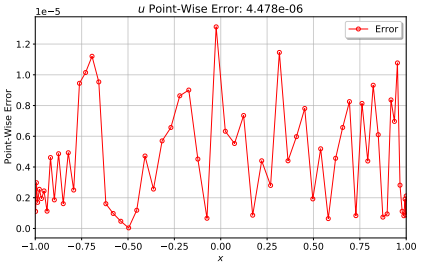
(A) Individual Losses



(B) Training Loss



(C) Predicted solution $\widehat{u}$



(D) Point-wise Error

FIGURE 4. Numerical simulations of a trained model (16) with $k_u = 3.5$ are presented using 10,000 solutions in the training set with 32 collocation points. The individual losses for $\mathcal{L}_u$ and $\mathcal{L}_{wf}$ are plotted on a semi-log plot over 100,000 epochs, but this model converged after approximately 65,000 epochs. The overall losses for the training and test sets can be seen on a semi-log plot, where we observe the test loss is *less than* the training loss. An out-of-sample predicted solution is plotted along with the known true solution with a mean relative ℓ^2 error on the order of 10^{-5}. A plot of the point-wise error is observed where the mean absolute error is on the order of 10^{-5}.

Hence, we define

$$LHS := \sum_{j=0}^{m} \left(-\int_{-1}^{1} \widehat{u}_x (\phi_j)_x + \int_{-1}^{1} k_u \widehat{u} \phi_j \right), \qquad RHS := \sum_{j=0}^{m} \left(\int_{-1}^{1} f \phi_j \right), \tag{19}$$

$$\mathcal{J}_{wf} := LHS - RHS,$$

then minimize $\mathcal{J}_{wf}$ together with the other errors. Here m stands for the number of test functions of the weak formulation, which will be used in the loss function. The loss function, $\mathcal{L}$, for our optimal model was

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_{wf}, \tag{20}$$

where $\mathcal{L}_u$ and $\mathcal{L}_{wf}$ as defined in (7).

3.2.1. Experimental Results. The Helmholtz equation is a linear wave equation, and we have found that the optimal neural architecture for solving it is also

entirely linear. We employ the same architecture as depicted in Figure 2a to achieve results similar to those obtained using the CDE. Surprisingly, a purely linear neural architecture, without any nonlinear activation functions, demonstrates the ability to generalize well from a small training set to a larger test set. To achieve this, we train the model for a considerable number of epochs, continuing until the model converges or until resources are required for other projects. In this context, we define a 'converged' model as one where the loss function $\mathcal{L}$ has reached convergence, meaning that the loss at a given epoch remains the same in subsequent epochs, persisting indefinitely.

A linear model, consisting of 10,000 in-sample solutions and trained using 64 collocation points, was trained for 100,000 epochs. The results are depicted in Figure 4. Throughout the training process, the loss values for the predicted solution $\widehat{u}$ and its associated weak form were recorded at each epoch, as shown in Figure 4a, illustrating their convergence. To evaluate the model's performance, testing was conducted using a separate 1,000 out-of-sample data set, as depicted in Figure 4b. Notably, both the training and test sets exhibit a smooth convergence, indicating that our model generalizes well to out-of-sample data despite having a small input training set. Figure 4c presents a comparison between the predicted solution $\widehat{u}$ and the true solution generated using the SEM (Spectral Element Method). The relative ℓ^2-error between them is measured to be 7.12×10^{-5}, indicating a relatively accurate prediction.

3.2.2. Discussion. Linear equations can be effectively solved using linear architectures. Employing linear architectures offers several benefits, including reduced computational complexity and the ability to generalize from a small training data set to a larger out-of-distribution data set. All linear models, depicted in Figure 2a, were constructed with a kernel size of 5, stride of 1, padding of 2, and 32 filters per channel. The LGNet algorithm provides accurate solutions to Equations (8) and (16) with various types of boundary conditions. Specifically, we use homogeneous Dirichlet boundary conditions for Equation (8) and homogeneous Neumann boundary conditions for Equation (16). Notably, the performance of each model remains relatively consistent regardless of the boundary conditions employed.

4. Non-linear and two-dimensional model

The performance of our neural network when applied to linear differential equations is accurate, especially when compared to other scientific machine learning models. However, in this section, we shift our focus to nonlinear models. Solving nonlinear differential equations poses significant challenges, and it would not be sufficient to use a linear neural network. Our neural architecture should reflect the nature of the differential equation itself. In other words, we aim to employ an architecture with nonlinear operations to effectively represent the solution of a nonlinear equation. In this section, we will introduce nonlinear neural architectures with various activation functions to explore the accuracy of trained models. Additionally, we will extend our analysis to two-dimensional problems, which naturally expand upon the foundation established by the one-dimensional model.

4.1. Burgers Equation. This problem can be further investigated by applying our algorithm to the canonical problem known as Burgers equation; a differential

equation known for its nonlinear shockwave characteristics. We define Burgers equation as

$$\begin{cases} -\varepsilon u_{xx} + uu_x = f, \\ g(\pm 1) = 0, \end{cases} \tag{21}$$

where ε is a diffusion coefficient. We utilize a uniform distribution when we generate random coefficients on the external forcing function f, that is our input data:

$$f = (3 + \upsilon_1)\sin[(1 + \upsilon_2)\pi x] + (3 + \upsilon_3)\cos[(1 + \upsilon_4)\pi x], \tag{22}$$

with $\upsilon_i \in \text{Uniform}[0, 2]$ for $1 \le i \le 4$. To address the issue of poor performance resulting from a training data set with a large variance, we have implemented a normalization step. Specifically, we normalize our input training set to have a mean of 0 and a standard deviation of 1. This approach, similar to the data pre-processing technique utilized in [4], helps enhance the performance of our network. Data pre-processing is a commonly employed technique in machine learning to improve network performance. For further information on data pre-processing, you can refer to sources such as [5] or [41].

Since homogeneous Dirichlet boundary conditions are used, the global basis functions for (21) are the same as in (10). We remark that the proposed algorithm can compute other boundary conditions, such as Neumann or Robin boundary conditions, since the Legendre basis provides an exact representation of the boundary conditions. By multiplying each side of Equation (21) by a test function, ϕ_k, and integrating from -1 to 1, we can derive the weak form, as explained earlier:

$$\begin{aligned} -\varepsilon \int_{-1}^{1} u(\phi_k)_{xx}\, dx + \frac{1}{2}\int_{-1}^{1} (u^2)_x(\phi_k)\, dx &= \int_{-1}^{1} f\phi_k\, dx \\ \Rightarrow \varepsilon \int_{-1}^{1} u_x(\phi_k)_x\, dx - \frac{1}{2}\int_{-1}^{1} u^2(\phi_k)_x\, dx &= \int_{-1}^{1} f\phi_k\, dx. \end{aligned} \tag{23}$$

We again set the residual as

$$LHS := \sum_{j=0}^{m} \left(\varepsilon \int_{-1}^{1} u_x(\phi_j)_x\, dx - \frac{1}{2}\int_{-1}^{1} u^2(\phi_j)_x\, dx \right), \quad RHS := \sum_{j=0}^{m} \left(\int_{-1}^{1} f\phi_j\, dx \right), \tag{24}$$

$$\mathcal{J}_{wf} := LHS - RHS.$$

Burgers' equation, being nonlinear, presents an intriguing challenge when utilizing neural networks to solve differential equations. To create a training data set, we employed the Picard iteration method with a tolerance level of 10^{-9}. Unlike linear data sets, nonlinear data sets have less stringent requirements in terms of training set data. Nevertheless, it is important to note that even the best linear models could not achieve the same level of accuracy. Hence, the chosen tolerance level of 10^{-9} is appropriate for training set purposes.

4.2. Two-dimensional convection-diffusion equations. We conducted a numerical simulation on the partial differential equations, which is the two-dimensional convection-diffusion equations (2D CDEs):

$$\begin{aligned} -\epsilon \Delta u + \mathbf{v} \cdot \nabla u &= f, \quad x \in \Omega, \\ u &= 0, \quad x \in \partial\Omega, \end{aligned} \tag{25}$$

where $\Omega = (-1,1)\times(-1,1)$ and $\mathbf{v} \in \mathbb{R}^2$ is a constant coefficient. We shall extend the one-dimensional Legendre-Galerkin results demonstrated in Section 3. By using the tensor product of one-dimensional basis functions, we propose multi-dimensional basis functions. In other words, let $\{\phi_k\}_{k=0}^{N-1}$ be a set of basis functions as in (3), we construct the two-dimensional numerical solution by approximating the function as a sum of basis functions:

$$u(x,y) \simeq \sum_{i,j=0}^{N-1} \alpha_{i,j}\phi_i(x)\phi_j(y). \tag{26}$$

Since the homogeneous Dirichlet boundary conditions are imposed in (25), we have $a_k = 0$ and $b_k = -1$ in (3). For given f, we develop our LGNet algorithm to find the coefficient $\{\widehat{\alpha}_{i,j}\}$ from which we can reconstruct the corresponding predicted numerical solution. If the random input dataset is given by

$$f(x,y) = m_1\cos(\pi w_1 x + \pi w_2 y) + m_2\sin(\pi w_3 x + \pi w_4 y), \tag{27}$$

where m_l and w_s for $1 \le l \le 2$ and for $1 \le s \le 4$ and follow normal distribution, then the corresponding output becomes

$$\{\widehat{\alpha}_{i,j}\} \implies \widehat{u}(x,y) = \sum_{i,j=0}^{N-1} \widehat{\alpha}_{i,j}\phi_i(x)\phi_j(y). \tag{28}$$

The set of global basis functions, $\phi_{i,j} = \phi_i(x)\phi_j(y)$, are then used as test functions in the weak formulation such that

$$-\epsilon\int_\Omega \nabla\hat{u}\nabla\phi_{i,j} + \int_\Omega \mathbf{v}\cdot\nabla\hat{u}\phi_{i,j} = \int_\Omega f\phi_{i,j}. \tag{29}$$

Hence, we set the residual

$$\begin{aligned} LHS &:= \sum_{i,j=0}^{m}\left(-\epsilon\int_\Omega \nabla\hat{u}\nabla\phi_{i,j} + \int_\Omega(\mathbf{v}\cdot\nabla\hat{u})\phi_{i,j}\right), \\ RHS &:= \sum_{i,j=0}^{m}\left(\int_\Omega f\phi_{i,j}\right), \\ \mathcal{J}_{wf} &:= LHS - RHS, \end{aligned} \tag{30}$$

then minimize $\mathcal{J}_{wf}$ together with $\mathcal{J}_u$ as before. Here m stands for the number of test functions of the weak formulation.

4.3. Experimental Results. Initially, we employ a neural architecture similar to the one found in [4, 49], as shown in Figure 2b and referred to as `NetA`. While current state-of-the-art neural networks, such as deep residual networks (ResNet), tend to perform better with increased network depth, we found that ResNet did not yield satisfactory results when using either Group Norm or Batch Norm [13, 18, 45]. However, we observed that the `NetA` architecture style was optimal for the LGNet. Each convolutional layer in this architecture has 32 filters, learned with a kernel size of 5, a stride of 1, and a padding of 2. To increase the depth of the network, we add additional blocks denoted as $\mathbb{B}$.

As a point of reference, we also implemented the sigmoid activation function as a drop-in replacement for ReLU for comparison purposes. This alternative neural architecture is denoted as `NetB`, and its activation function can be seen in Figure

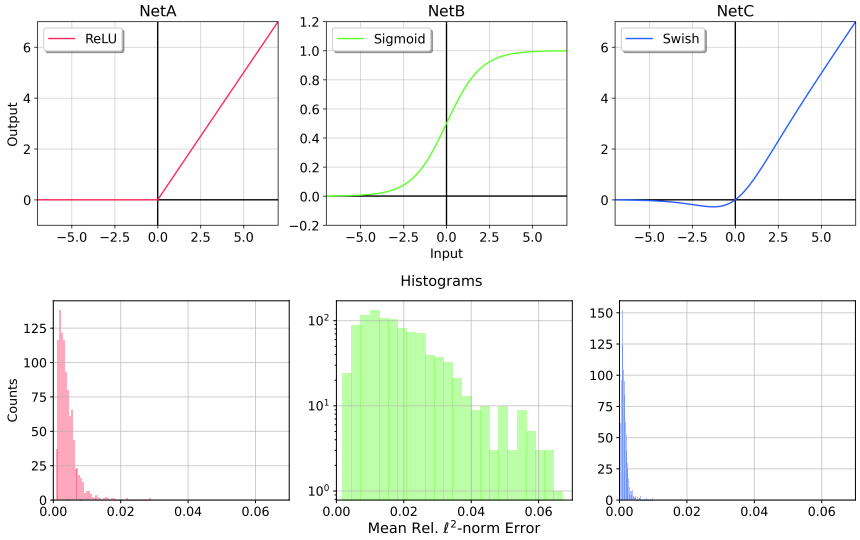


FIGURE 5. We investigate the effect that different activation functions have on the performance of LGNet. (Top) The activation function for each network is plotted. (Bottom) The relative ℓ^2-norm error histogram for each sample in the test dataset is recorded and a histogram plot is shown below its activation function. Each model is trained with an input training set of 15,000 solutions, 64 collocation points, 32 filters, 5 kernels, and 5 convolutional blocks. Models trained with the *Swish* activation function consistently outperform models using ReLU or sigmoid.

5. However, we found that ReLU consistently outperformed sigmoid by an order of magnitude. Saturation effects associated with sigmoid have been discussed in the literature, but in our case, we simply observed poor performance compared to ReLU.

Recent work has demonstrated the effectiveness of the '*Swish*' activation function; see Figure 5 [31]. This activation function has shown consistent improvement over the ReLU activation function in the application of CNNs to many different data sets which reflected in Figure 6. We use this as a drop-in replacement in our `NetA` architecture, for comparison, and refer to this network as `NetC`. The *Swish* activation function is given by

$$g(x) := x \cdot \text{sigmoid}(x) = \frac{x}{1 + e^{-x}}. \tag{31}$$

The swish function, defined by Equation (31), is a non-monotonic one-sided unbounded smooth function. Unlike the sigmoid activation function used in `NetB`, the swish function does not experience saturation, which is a drawback of the sigmoid function. Additionally, the swish activation function provides both negative and positive activation values, which is a characteristic that ReLU does not possess.

An example of a trained `NetC` model can be observed in Figure 7. Our best model was configured with 32 filters, a kernel size of 5, and 10 blocks. To validate

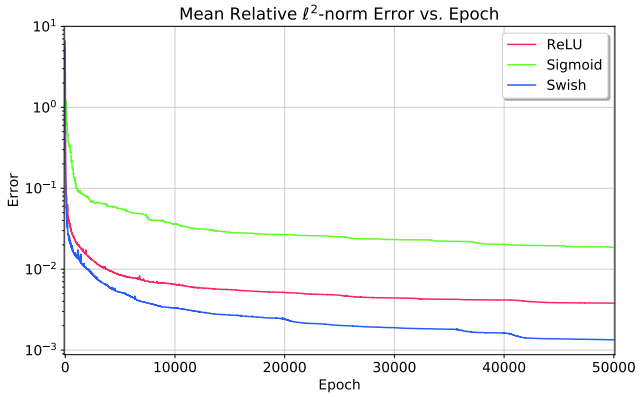
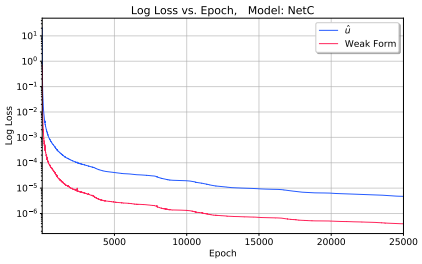


FIGURE 6. A comparison of different activation functions can be seen in the mean relative ℓ^2-norm error vs. epoch plot above; each model has the same parameters as Figure 5. The error is computed on the test data set and recorded at each epoch. The *Swish* activation function function consistently outperforms both sigmoid and ReLU. Both ReLU and *Swish* are an order of magnitude better than sigmoid, on average. The performance of each model is a reflection of the histograms in Figure 5.

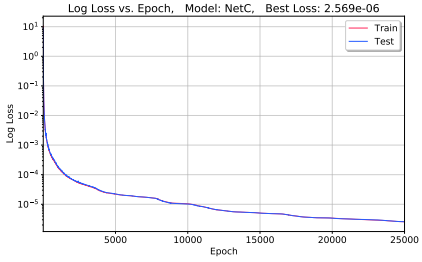
the model's performance, we conducted cross-validation by comparing it with out-of-sample data at each epoch, as depicted in Figure 7b. We employed the weak form for regularization, and the corresponding loss values are presented in Figure 7a. For an out-of-sample forcing function, $\tilde{f}$, the predicted solution, $\widehat{u}$, exhibited a relative ℓ^2-norm error of approximately 7.1×10^{-4}, while the average point-wise error for this solution was around 2×10^{-4}, as shown in Figure 7d. Furthermore, the mean relative ℓ^2-norm error on an out-of-sample dataset comprising 1,000 solutions was found to be 8.8×10^{-4}.

As a natural extension, we have implemented the two-dimensional equations. In Figure 8, we present the numerical solution and exact solution of (25). The numerical solution is generated using the 2D LGNet introduced in Section 4.2. For the experiment, we used 32 filters and a kernel size of 5. The model was trained for 50,000 epochs with 10,000 in-sample solutions and a 16×16 collocation point grid. Figure 8 illustrates a predicted solution, $\hat{u}$, compared to the true solution, u, obtained using the SEM. The relative l^2-error between the predicted and true solutions is approximately 4×10^{-4}.

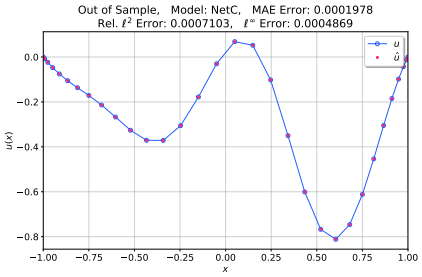
We tabulated the numerical results for each equation in Table 1. For CDE we used a linear model and let the model train for 100,000 epochs. The model achieved a mean relative ℓ^2-norm error on the order of 10^{-6}. Similarly, we used a linear architecture for the Helmholtz equation and achieved a mean relative ℓ^2-norm error on the order of 10^{-5}. The nonlinear model for Burgers' equation utilized the `NetC` architecture with 25,000 epochs. This model was able to achieve a mean



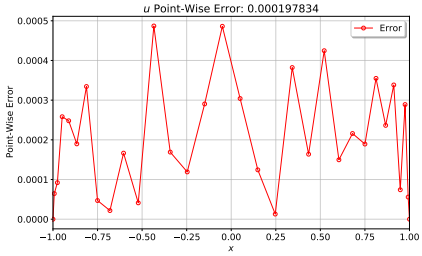
(A) Individual Losses



(B) Training loss



(C) Predicted solution $\widehat{u}$



(D) Point-wise Error

FIGURE 7. Numerical examples of the model (21) with $\varepsilon = 0.5$ are displayed using 10,000 solutions in the training set with 32 collocation points. The individual losses for $\mathcal{L}_u$ and $\mathcal{L}_{wf}$ are plotted on a semi-log plot and the overall losses for the training and test sets can be seen on a semi-log plot. An out-of-sample predicted solution is plotted along with the known true solution with a mean relative ℓ^2 error on the order of 10^{-4}. A plot of the point-wise error is observed where the average mean absolute error is on the order of 10^{-4}.

TABLE 1. The accuracy of trained models is presented here. For each equation, the parameters used in training are tabulated for clarity. Each model can achieve a mean relative ℓ^2-norm error of $\mathcal{O}(10^{-5})$ when using the L-BFGS optimizer. Data augmentation is used on nonlinear input data to realize comparable mean relative ℓ^2-norm error. Thus, effectively gaining comparable performance on nonlinear equations as linear equations.

Equation	Model	Epochs	blocks	filters	ks	Mean Rel. ℓ^2 Error
CDE	Linear	100,000	-	32	5	1.26×10^{-6}
Helmholtz	Linear	100,000	-	32	5	4.57×10^{-5}
Burgers	`NetC`	25,000	4	32	5	8.81×10^{-4}
2D CDE	`NetC`	50,000	1	32	5	4.37×10^{-4}

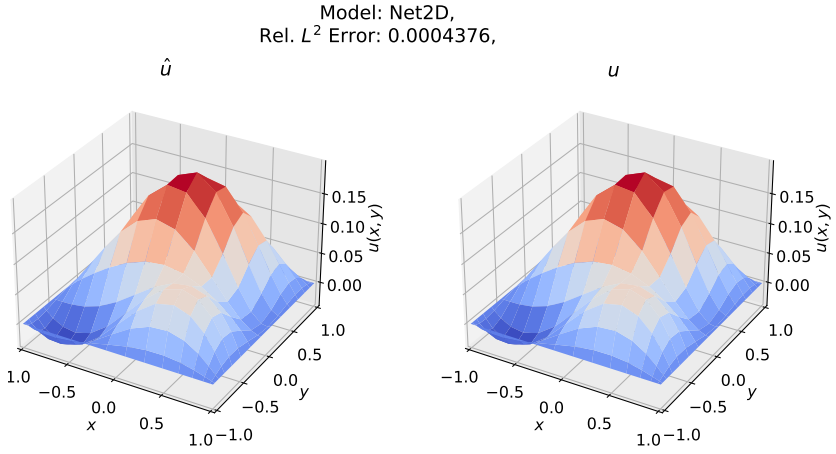


FIGURE 8. Comparison of the numerical solution $\hat{u}$ of (25) implemented by the two-dimensional LGNet (left panel) after 25,000 epochs and the exact solution u of (25) (right panel).

relative ℓ^2-norm error on the order of 10^{-4}. Lastly, the 2D CDE model achieved a mean relative ℓ^2-norm error on the order of 10^{-4}.

4.4. Discussion. The statistical nature of the neural architectural model introduces more room for error when training NNs, if the variance of the input data set is too large, but still yields sufficiently accurate results. The L-BFGS algorithm has its benefits during full-batch gradient descent, but when the size of the data set exceeds the memory limitations of the GPU, mini-batch L-BFGS gradient descent can diverge. Also, introducing a pooling operation into the neural architecture could greatly improve upon local invariance, but we have not measured the effects here.

The performance of a DNN to predict solutions of Equation (21) can yield comparable results to Equations (8) or (16), but only with data augmentation. If the input forcing function, f, in Equation (22) is normalized prior to training then nonlinear equations can enjoy an accuracy comparable to that of linear equations. We also notice that the (linear) two-dimensional equations demonstrate comparable results to the one-dimensional (linear) equations.

Remark 4.1. *The current architecture can be extended to time dependent problems, a PDE model. For instance, we consider a time dependent Burgers' equations,*

$$\begin{aligned} u_t - u_{xx} + uu_x &= f, \quad x \in (-1,1), \\ u(\pm 1) &= 0. \end{aligned} \tag{32}$$

Applying the backward Euler method in time, the equation $(32)_1$ *becomes*

$$\frac{u^{n+1} - u^n}{\Delta t} - u_{xx}^{n+1} + u^{n+1}u_x^{n+1} = f^{n+1}. \tag{33}$$

Hence, by setting $v := u^{n+1}$ *we deduce that*

$$-v_{xx} + (v^2)_x + \frac{1}{\Delta t} v = f^{n+1} + \frac{1}{\Delta t} u^n. \tag{34}$$

However, the structure of (34) *is very similar to that in Section 4, and a similar approach/architecture of Section 4 can be used. It is noteworthy to mention that we need to build only one network for all time steps. Indeed, the only difference in different time steps is* u^n *in* (34)*, which is our input data. Hence, one neural network is enough to solve the time-dependent model. Further discussion will be provided in the forthcoming article.*

5. Conclusion

We develop deep neural networks to learn the coefficients associated with global basis functions, enabling accurate reconstruction of the numerical solution. By imposing the weak form of the governing equations, we improve the accuracy of the predicted solutions. This methodology can be further extended to time-dependent partial differential equations by performing time discretization or to higher-dimensional problems by employing higher-dimensional convolutional kernels. Moreover, with a simple modification of the LGNet framework, it is possible to utilize different orthogonal polynomial basis functions. The versatility of LGNet allows for the prediction of a wide range of solutions to various PDEs with different boundary conditions.

Another intriguing application for the LGNet is the study of stiff partial differential equations. Convection-dominated singularly perturbed problems pose significant challenges due to the presence of sharp transitions within thin layers caused by the small diffusive parameter ϵ. In a previous work [6], the authors explored an enriched spectral method for solving such singularly perturbed models. This approach involved augmenting the Legendre basis functions with analytically-determined boundary layer elements called *correctors* to capture the complex behavior near the boundary. With our architecture's flexibility, we can introduce an additional basis function with a learning parameter, resulting in an approximation of the form $u(x) \approx \sum_{k=0}^{N-1} \alpha_k \phi_k(x) + \alpha_N \phi_N(x)$, where ϕ_k (for $0 \le k \le N-1$) represents the standard Legendre basis and ϕ_N corresponds to the enriched basis generated by the corrector function. This example highlights the adaptability of our LGNet algorithm, which will be further discussed in an upcoming article.

Statements and Declarations

The authors declare that they have no competing interests.

Acknowledgements

The work of Youngjoon Hong was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A2C1093579) and Korean Government (MSIT) (2022R1A4A3033571).

References

[1] Keith Rudd, Silvia Ferrari. A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. Neurocomputing, 155:277 – 285, 2015.

[2] David M. Allen. The relationship between variable selection and data agumentation and a method for prediction. Technometrics, 16(1):125–127, 1974.
[3] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units, 2016.
[4] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. Proceedings of the National Academy of Sciences, 116(31):15344C15349, Jul 2019.
[5] Christopher M. Bishop. Pattern recognition and machine learning. Springer, 2006.
[6] Mickael D. Chekroun, Youngjoon Hong, and Roger M. Temam. Enriched numerical scheme for singularly perturbed barotropic quasi-geostrophic equations. Journal of Computational Physics, 416:109493, 2020.
[7] Zhen Chen and Dongbin Xiu. On generalized residual network for deep learning of unknown dynamical systems. Journal of Computational Physics, 438:110362, 2021.
[8] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification, 2016.
[9] George Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2:303–314, 1989.
[10] Philip J. Davis and Philip Rabinowitz. Methods of numerical integration. Dover Publications, Inc., Mineola, NY, 2007. Corrected reprint of the second (1984) edition.
[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.
[12] David Gottlieb and Steven A. Orszag. Numerical Analysis of Spectral Methods. Society for Industrial and Applied Mathematics, 1977.
[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
[14] Youngjoon Hong, Chang-Yeol Jung, and Roger Temam. On the numerical approximations of stiff convection-diffusion equations in a circle. Numer. Math., 127(2):291–313, 2014.
[15] Youngjoon Hong and David P. Nicholls. A high-order perturbation of surfaces method for scattering of linear waves by periodic multiply layered gratings in two and three dimensions. J. Comput. Phys., 345:162–188, 2017.
[16] Youngjoon Hong and David P. Nicholls. A high-order perturbation of surfaces method for vector electromagnetic scattering by doubly layered periodic crossed gratings. J. Comput. Phys., 372:748–772, 2018.
[17] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359 – 366, 1989.
[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.
[19] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations, 2019.
[20] Byungjoo Kim, Bryce Chedomelka, Jinyoung Park, Jaewoo Kang, Youngjoon Hong, and Hyunwoo J Kim. Robust neural networks inspired by strong stability preserving runge-kutta methods. In ECCV, 2020.
[21] Yoon Kim. Convolutional neural networks for sentence classification, 2014.
[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.
[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278C2324, 1998.
[24] Zongyi Li, Hongkai Zheng, Nikola Borislavov Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Andrew Stuart, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations, 2022.
[25] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence, 3(3):218–229, 2021.

[26] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. SIAM Review, 63(1):208–228, 2021.
[27] Olalekan Ogunmolu, Xuejun Gu, Steve Jiang, and Nicholas Gans. Nonlinear systems identification using deep dynamic neural networks, 2016.
[28] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686 – 707, 2019.
[29] Maziar Raissi and George Em Karniadakis. Hidden physics models: machine learning of nonlinear partial differential equations. J. Comput. Phys., 357:125–141, 2018.
[30] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 367(6481):1026–1030, 2020.
[31] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.
[32] S. C. Reddy and J. A. C. Weideman. The accuracy of the Chebyshev differencing method for analytic functions. SIAM J. Numer. Anal., 42(5):2176–2187, 2005.
[33] Viktor Reshniak and Clayton G. Webster. Robust learning with implicit residual networks. Machine Learning and Knowledge Extraction, 3(1):34–55, 2021.
[34] Jon A. Rivera, Jamie M. Taylor, ngel J. Omella, and David Pardo. On quadrature rules for solving partial differential equations using neural networks. Computer Methods in Applied Mechanics and Engineering, 393:114710, 2022.
[35] Jie Shen, Tao Tang, and Li-Lian Wang. Spectral methods, volume 41 of Springer Series in Computational Mathematics. Springer, Heidelberg, 2011. Algorithms, analysis and applications.
[36] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, 375:1339–1364, 2018.
[37] E.G. Steward. Fourier Optics: An Introduction (Second Edition). Dover Books on Physics. Dover Publications, 2004.
[38] Stinchcombe and White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In International 1989 Joint Conference on Neural Networks, pages 613–617 vol.1, 1989.
[39] Wensi Tang, Guodong Long, Lu Liu, Tianyi Zhou, Jing Jiang, and Michael Blumenstein. Rethinking 1d-cnn for time series classification: A stronger baseline, 2020.
[40] Robert Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 58(1):267–288, 1996.
[41] Vladimir Naumovich. Vapnik. The nature of statistical learning theory. Springer, 2000.
[42] Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J. Osher. Enresnet: Resnets ensemble via the feynman–kac formalism for adversarial defense and beyond. SIAM Journal on Mathematics of Data Science, 2(3):559–582, 2020.
[43] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. Science Advances, 7(40):eabi8605, 2021.
[44] E. Weinan and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6:1–12, 2017.
[45] Yuxin Wu and Kaiming He. Group normalization, 2018.
[46] Xuping Xie, Clayton Webster, and Traian Iliescu. Closure learning for nonlinear model reduction using deep residual neural network. Fluids, 5(1), 2020.
[47] Rui Xu, Dongxiao Zhang, Miao Rong, and Nanzhe Wang. Weak form theory-guided neural network (tgnn-wf) for deep learning of subsurface single- and two-phase flow. Journal of Computational Physics, 436:110318, 2021.
[48] Yulei, Liao, , 14603, , Yulei Liao, Pingbing, Ming, , 10035, , and Pingbing Ming. Deep nitsche method: Deep ritz method with essential boundary conditions. Communications in Computational Physics, 29(5):1365–1384, 2021.
[49] Jiawei Zhuang, Dmitrii Kochkov, Yohai Bar-Sinai, Michael P. Brenner, and Stephan Hoyer. Learned discretizations for passive scalar advection in a 2-d turbulent flow, 2020.

Department of Mathematics and Statistics, San Diego State University, San Diego, CA, USA
E-mail: brycechudomelka@gmail.com

Department of Mathematical Sciences, KAIST, Daejeon, Korea
E-mail: hongyj@kaist.ac.kr

Department of Mathematics and Statistics, San Diego State University, San Diego, CA, USA
E-mail: jmorgan4844@sdsu.edu

Department of Computer Science, Korea University, Seoul, Korea
E-mail: hyunwoojkim@korea.ac.kr and lpmn678@korea.ac.kr