

POLYNOMIAL-AUGMENTED NEURAL NETWORKS (PANNs) WITH WEAK ORTHOGONALITY CONSTRAINTS FOR ENHANCED FUNCTION AND PDE APPROXIMATION *

MADISON COOLEY*, SHANDIAN ZHE*, ROBERT M. KIRBY*, AND VARUN SHANKAR*

Abstract. We present polynomial-augmented neural networks (PANNs), a novel machine learning architecture that combines deep neural networks (DNNs) with a polynomial approximant. PANNs combine the strengths of DNNs (flexibility and efficiency in higher-dimensional approximation) with those of polynomial approximation (rapid convergence rates for smooth functions). To aid in both stable training and enhanced accuracy over a variety of problems, we present (1) a family of orthogonality constraints that impose mutual orthogonality between the polynomial and the DNN within a PANN; (2) a simple basis pruning approach to combat the curse of dimensionality introduced by the polynomial component; and (3) an adaptation of a polynomial preconditioning strategy to both DNNs and polynomials. We test the resulting architecture for its polynomial reproduction properties, ability to approximate both smooth functions and functions of limited smoothness, and as a method for the solution of partial differential equations (PDEs). Through these experiments, we demonstrate that PANNs offer superior approximation properties to DNNs for both regression and the numerical solution of PDEs, while also offering enhanced accuracy over both polynomial and DNN-based regression (each) when regressing functions with limited smoothness.

Key words. Scientific Machine Learning, Polynomial Methods, Physics-Informed Neural Networks, Partial differential equations

MSC codes. 68T07, 68U99, 65N99

1. Introduction. Recent advancements in machine learning, particularly within deep neural networks (DNNs), have significantly impacted various scientific fields due to their broad applicability and flexibility [34, 38, 74]. DNNs are popular primarily due to their expressiveness, scalability, and efficient optimization with gradient descent methods through the use of automatic differentiation. DNNs are versatile tools capable of solving diverse problems ranging from classification and regression to PDE approximation and image recognition. Recently, DNNs have also been applied to both forward and inverse partial differential equations (PDEs) in the form of physics-informed neural networks (PINNs), which extend the capabilities of standard DNNs by incorporating a physics-based loss term into the data loss [61, 62]. DNNs are also generalizable on diverse data and domain types without requiring *a priori* knowledge of solution characteristics [48, 50]. This property is especially beneficial in the context of PINNs, as they eliminate the need for mesh generation that is mandated by many traditional numerical methods for PDEs. Furthermore, DNNs arguably break the curse of dimensionality [7, 12, 29, 32], meaning that as the problem dimension increases, the required network size for accurate approximations does not grow exponentially with dimension. These traits allow DNNs to approximate complicated functions effectively [18].

Despite the many advantages of DNNs, their use comes with significant challenges in model initialization and training [30, 39]. For instance, DNNs encounter issues like vanishing or exploding gradients during training, where the back-propagated gradients either approach zero or increase exponentially respectively; this could either result in very slow training or high generalization errors [28, 57]. Spectral bias is an additional

*

Funding: This work is funded in part by the Army Research Office (ARO).

†University of Utah (mcooley@cs.utah.edu, zhe@cs.utah.edu, kirby@cs.utah.edu, shankar@cs.utah.edu).

challenge, manifesting as quick convergence to the low-frequency components of the target solution while struggling with high-frequency components [5, 60, 72]—an issue that also extends to PINNs [70]. Furthermore, DNNs are prone to overfitting the training data [69], hence compromising their generalizability.

Traditional approximation methods, especially those involving polynomials, remain a strong choice for both function approximation and the solution of PDEs. However, polynomial least-squares methods, while robust in many applications, also face many challenges. Polynomial least-squares typically require oversampling to achieve stability [3, 2, 45, 51] even on tensor-product grids. Polynomial approximation can also be generalized to non-tensor-product grids (and hence irregular domains), but this requires the use of sophisticated techniques such as on-the-fly basis function recomputation [9, 19, 73] or localization [11, 33, 68]. In addition, naive polynomial approximation is subject to the curse of dimensionality, where the number of polynomial basis functions grows exponentially with dimension. Common techniques to combat this explosive growth of the number of basis functions include compressive sensing [1, 44] (which induces sparsity in the polynomial coefficients), Smolyak/sparse grids (which utilize sparse tensor-product grids) [10, 26, 40, 71], or hyperbolic cross approximation (which utilizes only a subset of the polynomial basis) [20, 21, 67]. In general, (global) polynomial methods are well-suited to approximating smooth target functions, while DNNs often perform better approximating non-smooth functions [18, 22], at least partly due to their connections to piecewise polynomial approximation [54, 55].

Motivated by these observations, we introduce Polynomial-Augmented Neural Networks (PANNs), which combine the strengths of both DNNs and polynomials. Specifically, we augment a standard DNN with a preconditioned polynomial layer containing trainable coefficients, and mutually optimize the two approximations using a novel family of eight orthogonalization constraints that enforce weak orthogonality between the polynomial and DNN bases; we also precondition the DNN itself. While a naive addition of a polynomial layer can re-introduce the curse of dimensionality into this augmented approximation, we leverage basis truncation to control the number of polynomial basis terms for increasing dimension and high polynomial degrees. From the DNN perspective, the PANN architecture can be viewed as a residual block with a set of transformed skip connections containing trainable strength connection parameters.

We show in this work that the resulting PANN architecture significantly improves DNN approximations of polynomial target functions (unsurprisingly). More importantly, we present empirical results showing that the PANN architecture is superior to either DNNs or polynomials on tasks such as approximating functions with finite smoothness, high-dimensional function approximation, and approximating noisy functions drawn from a high-dimensional housing dataset. Further, when PANNs are used as physics-informed networks for the solution of PDEs (PI-PANNs), we observe relative ℓ_2 errors that are orders of magnitude lower than traditional PINNs. We also show that the choice of orthogonality constraint can affect approximation quality and wall-clock training times in an application-dependent fashion.

Other work has explored hybrid approximation techniques. For instance, Π -Nets, introduced in [14, 15], involve modifying convolutional neural nets to output polynomials of the input variables, represented via high-order tensors. In contrast, our approach outputs a linear combination of DNNs and polynomials. There are also many other works that directly replace DNNs with polynomials [13, 27, 35, 43]. Modern radial basis function-finite difference (RBF-FD) methods combine RBFs and polynomials together with orthogonality constraints that enforce polynomial reproduction.

There, since closed-form expressions are available for the RBF basis, no training is required and the orthogonality constraint is enforced by treating the polynomial coefficients as Lagrange multipliers [4, 6, 23, 63, 65]. RBF-FD is primarily used to generate finite difference weights on scattered points, but barring the freedom to handle irregular point sets and domains, does suffer from many of the same issues as traditional polynomial approximation (for instance, the curse of dimensionality and difficulties tackling finitely-smooth function data). In contrast, PANNs are global approximators that must be trained; the presence of a DNN introduces an entire family of orthogonality constraints, each presenting different cost-accuracy tradeoffs on different problems, but the overall method is robust to noise and inherits the benefits of DNNs.

The remainder of this paper is structured as follows: [section 2](#) provides essential background and notation. The new PANN architecture, its training, preconditioning, and the new orthogonality constraints are described in [section 3](#). Then [section 4](#) presents our numerical experiments and findings, including an assessment of computational cost and accuracy compared to baseline methods. Finally, [section 5](#) discusses the results and outlines future research directions.

2. Background. In this section, we define the general optimization problems we are interested in, along with a brief review of DNNs and certain classes of polynomial approximation methods. The problem dimension is denoted by d , and ℓ signifies the polynomial degree used to generate the polynomial bases. The total number of training points is represented by n , and w refers to the width of the last layer of a DNN. The total number of polynomial bases, which is also the width of the polynomial layer, is denoted by m . The DNN basis coefficients are symbolized by a_j for $j = 1, \dots, w$, while the polynomial layer basis coefficients are represented by b_k for $k = 1, \dots, m$. The DNN basis functions are indicated by ψ_j for $j = 1, \dots, w$, and the polynomial layer basis functions are represented by ϕ_k for $k = 1, \dots, m$. In this paper, we primarily focus on the supervised regression problem with the form,

$$(2.1) \quad \operatorname{argmin}_{\theta} \sum_{i=1}^{N_{data}} |u_{\theta}(x_i) - u(x_i)|^2,$$

for N_{data} training points $x \in \mathbb{R}^d$ such that u is the true solution we aim to approximate and u_{θ} is the model parameterized by its weights and biases θ . We additionally concentrate on semi-supervised approaches for solving partial differential equations (PDE) of the form,

$$(2.2) \quad \operatorname{argmin}_{\theta} \sum_{i=1}^{N_{data}} |u_{\theta}(x_i^b) - u(x_i^b)|^2 + \lambda \sum_{j=1}^{N_{PDE}} |\mathcal{F}[u_{\theta}](x_j^r) - f(x_j^r)|^2,$$

where \mathcal{F} is some (linear or non-linear) differential operator operating on N_{PDE} collocation points $\{x_i^r\}_{i=1}^{N_{PDE}}$ from the domain Ω and N_{data} points $\{x_i^b\}_{i=1}^M$ from the boundary of the domain ($\partial\Omega$). λ is a Lagrange multiplier that balances the learning between the data and residual loss terms. Subsequent sections detail extensions of [\(2.1\)](#) and [\(2.2\)](#), which incorporate custom orthogonalizing regularization terms, along with polynomial preconditioning.

2.1. Deep Neural Networks. Following the convention of [16], we represent the family of DNNs, $\mathcal{N} \in \mathbb{R}^d \rightarrow \mathbb{R}$ of width w , as a linear combination of adaptive

basis functions given by

$$(2.3) \quad \mathcal{N}(x; a, \theta^h) = \sum_{j=1}^w a_j \psi_j(x; \theta^h),$$

where each a_j for $j = 1, \dots, w$ and θ^h constitute the weights and biases in the last layer and hidden layers respectively, forming the set of all network parameters θ . Then, each ψ_j are non-linear activation functions such as ReLU or Tanh acting on the outputs of the hidden layers. The parameters θ are computed through some iterative optimization technique. In this work, we use variants of gradient descent methods such as ADAM [36] and L-BFGS [47].

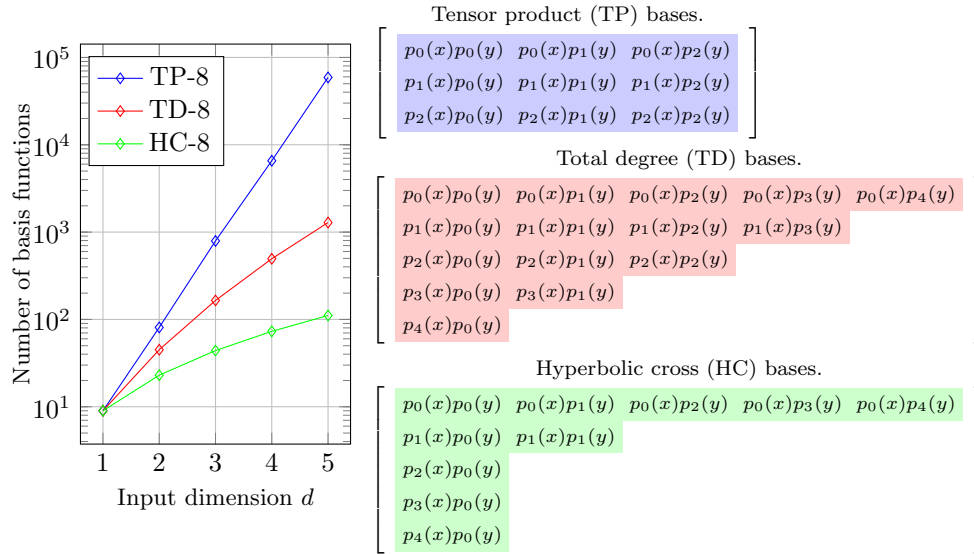


FIG. 2.1. Visual depiction of bases sets using different generation techniques on the right and the total number of basis function each method produces for increasing problem dimension.

2.2. Polynomial Methods. We define a family of polynomial models, $\mathcal{P} \in \mathbb{R}^d \rightarrow \mathbb{R}$, in a similar form to (2.3), as a linear combination of m orthogonal polynomials ϕ_k and parameters b_k . Specifically,

$$(2.4) \quad \mathcal{P}(x; b) = \sum_{k=1}^m b_k \phi_k(x),$$

where ϕ_k for $k = 1, \dots, m$, form a polynomial basis in d dimensions. Various approaches exist for basis generation, including tensor-product, total-degree, and hyperbolic-cross methods. Figure 2.1 illustrates each type of basis, comparing their cardinality to the function space spanned by each. Tensor-product bases grow exponentially by dimension, including many highly oscillatory basis functions. Insights from Smolyak cubature [59] suggest that the influence of highly oscillatory components diminishes with increased dimension, rendering tensor-product bases less computationally efficient. Conversely, the total-degree method constrains the combined degrees of each

basis to a certain threshold, thereby ensuring a more gradual increase in size while including more bases with low oscillation. Hyperbolic-cross bases exhibit the slowest growth concerning dimensionality, albeit at the expense of expressivity. Weighing these considerations, we prefer total-degree bases where the combined degree limit is ℓ . In this work, we employ total-degree Legendre polynomials combined with basis pruning to ensure a slower growth in the number of basis functions (as a function of spatial dimension), but our approaches carry over straightforwardly to other polynomial bases also.

3. Polynomial Augmented Neural Networks. This section outlines our enhanced neural network architecture incorporating polynomials, expanding on the preconditioning methods in [subsection 3.1](#), and our unique discrete orthogonality constraints in [subsection 3.2](#). The algorithmic framework, including the selection and truncation of polynomial bases, is detailed in [subsection 3.3](#).

We aim to strategically augment a standard DNN with structured polynomials containing trainable coefficients. [Figure 3.1](#) (left) illustrates our hybrid model, the **Polynomial-Augmented Neural Network** (PANN). We define the model’s prediction, u_θ , as the sum of the DNN’s output, $\mathcal{N}(x)$ (defined in [\(2.3\)](#)), and the output of the polynomial layer, $\mathcal{P}(x)$ (defined in [\(2.4\)](#)) as follows:

$$(3.1) \quad u_\theta(x) = \mathcal{N}(x) + \mathcal{P}(x) = \sum_{j=1}^w a_j \psi_j(x; \theta^h) + \sum_{k=1}^m b_k \phi_k(x),$$

In this paper, we primarily present PANNs through an adaptive basis viewpoint [\[16\]](#). However, one can interpret PANNs as a type of residual network such that the DNN output is combined with a set of polynomial transformed skip connections that have trainable strength parameters, visualized in right figure of [Figure 3.1](#). An intuition for the PANN architecture in [\(3.1\)](#) can be seen in [Figure 3.2](#), which compares the loss landscapes of a PANN used as a physics-informed neural network (PINN) and a standard PINN. For this experiment, we simply perturbed the two dominant eigenvectors (δ , ν) of the loss Hessian, and evaluated the adjusted loss \mathcal{L}' across a specified range for α and β such that, $\mathcal{L}'(\alpha, \beta) = \mathcal{L}(\theta + \alpha\delta + \beta\nu)$ and $\alpha, \beta \in [-\alpha_0, \alpha_0] \times [-\beta_0, \beta_0]$ [\[37, 46\]](#).

[Figure 3.2](#) clearly reveals that polynomial augmentation smooths the loss landscape, suggesting that it helps avoid local minima, hence simplifying the optimization process and possibly boosting model accuracy (which we verify in a later section). We use a single polynomial layer rather than a “deep” architecture; this was motivated, in part, by previous findings which demonstrate that depth in polynomial networks does not equate to enhanced representation capabilities (unlike in the case of DNNs with either piecewise polynomial or other nonlinear activation functions) [\[41\]](#). More importantly, our design benefits from the static nature of the polynomial bases, allowing us to precompute and store the bases and their derivatives for training efficiency—a detail we expand upon in [subsection 3.3](#).

3.1. Preconditioning. The convergence rates of many numerical methods rely on problem conditioning, where ill-conditioned problems typically exhibit slower convergence. Problem conditioning not only influences traditional methods but is also a critical factor in the training difficulties of DNNs, including PINNs [\[37\]](#). In this work, we apply the polynomial preconditioning techniques from [\[31, 53\]](#). This preconditioning technique was developed for least-squares approximation scenarios with extensive oversampling, *i.e.*, with far fewer function samples (n) than polynomial basis functions (m). This preconditioner also attempts to maximize sparsity in the poly-

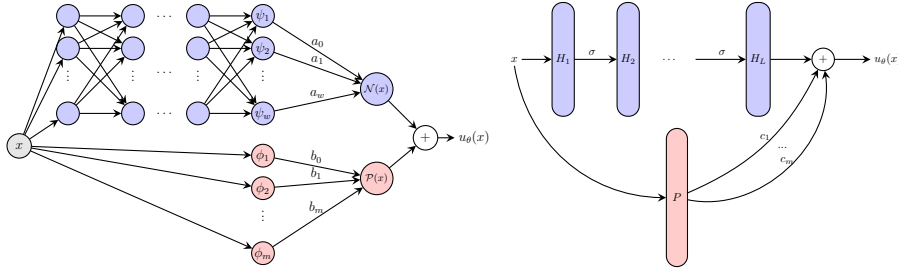


FIG. 3.1. Both figures show the proposed neural network architecture with polynomial layer (PANN). The left figure demonstrates the architecture from the adaptive basis viewpoint where each ψ_i and a_i for $i = 1, \dots, w$ are the DNN bases and coefficients, while ϕ_j and b_j for $j = 1, \dots, m$ are the polynomial layer bases and coefficients respectively. u_θ is the model output which is a linear combination of the DNN and polynomial layer bases and coefficients. Alternatively, the right figure demonstrates the architecture as a residual block with transformed skip connections such that each H_k for $k = 1, \dots, L$ represent the hidden layers of the DNN, σ are non-linear activations, P is the polynomial layer and c_j for $j = 1, \dots, m$ are the transformed and adaptive skip connections.

nomial coefficients. The optimization problem considered in [31] aims to solve the inequality-constrained l^1 -minimization problem defined as:

$$(3.2) \quad \underset{\theta}{\operatorname{argmin}} \|\theta\|_1 \quad \text{such that} \quad \|K\Phi b - Kf\|_2 \leq \epsilon.$$

Here, the diagonal matrix $K \in \mathbb{R}^{n \times n}$ is constructed to improve the l^1 -minimization problem's tractability and aids in recovering sparse solutions. Formally,

$$(3.3) \quad K_{n,n} = \sqrt{\frac{m}{\sum_{i \in \Lambda} \phi_i^2(x)}},$$

where Λ is the set of all polynomial bases. In the context of polynomial least-squares approximation, this preconditioning attempts to rescale polynomial bases with large norms, leading to a system where each basis contributes more equally.

In this work, we consider a more general form of (3.3) such that the objective function constrains both the preconditioned error norm and parameter norm. Further, we apply this preconditioning to both the DNN and the polynomial bases. The updated optimization problem considered is, therefore,

$$(3.4) \quad \min_{a,b,\theta} \|K\Psi a + K\Phi b - Kf\|_2^2 + \lambda_r \|\theta\|_1,$$

where λ_r is a prescribed Lagrange multiplier.

3.2. Discrete orthogonality constraints. We now present a novel family of orthogonality constraints designed to induce a “weak” orthogonality between the DNN basis and the polynomial layer within the PANN. This orthogonality was inspired by a philosophically similar approach utilized in modern radial basis function-finite difference (RBF-FD) methods [4, 24], where RBF expansions are computed in such a way that they are orthogonal to some polynomial bases (typically total-degree); this orthogonality constraint endows RBF-FD weights with polynomial reproduction properties, thereby controlling their convergence rates.

While deriving this constraint in RBF-FD methods is straightforward, it is significantly more challenging in the context of PANNs, which require training to determine

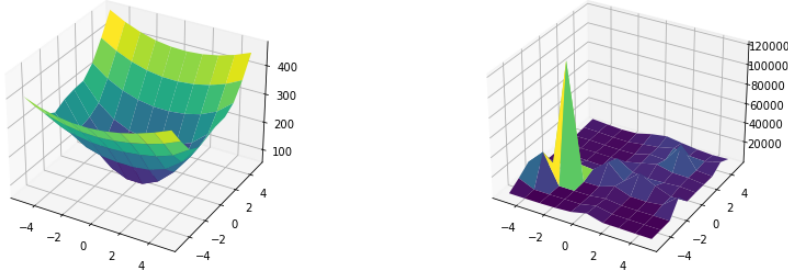


FIG. 3.2. Loss landscapes of a physics-informed PANN (left) and standard PINN (right) on a 2D Poisson problem.

not only polynomial coefficients, but DNNs coefficients *and* bases. Fortunately, enforcing orthogonality is nevertheless possible. For instance, a *continuous* orthogonality constraint between the DNN output $\mathcal{N}(x)$ and the polynomial layer $\mathcal{P}(x)$ can be expressed as $\int_{\Omega} \mathcal{N}(x)\mathcal{P}(x) = 0$. While it is generally impossible to compute this integral analytically, it can be approximated to arbitrary accuracy by a suitable quadrature formula provided the DNN is sufficiently smooth (or even continuous) [17]. However, this approach does not generalize straightforwardly to irregular domains or higher dimensional problems without sacrificing the meshless nature of DNNs.

Our approach involves replacing the continuous orthogonality constraint with discrete alternatives that obviate the need for quadrature. To see how, consider more carefully the continuous constraint $\int_{\Omega} \mathcal{N}(x)\mathcal{P}(x) = 0$. This constraint yields many equivalent forms, some of which include,

$$(3.5) \quad \int_{\Omega} \sum_{j=1}^w a_j \psi_j(x; \theta^h) \mathcal{P}(x) = 0 \iff \int_{\Omega} \sum_{k=1}^m b_k \phi_k(x) \mathcal{N}(x; \theta) = 0.$$

One straightforward way for these integrals to be zero is to enforce that the integrands themselves be zero. This can be enforced by forcing the summands to be zero, which in turn can be done by enforcing that each term in the summands be zero. This chain of reasoning leads to two distinct families of discrete constraints:

$$(3.6) \quad a_j \psi_j(x) \mathcal{P}(x) = 0, \quad j = 1, \dots, w,$$

$$(3.7) \quad b_k \phi_k(x) \mathcal{N}(x) = 0, \quad k = 1, \dots, m.$$

It is important to note that the discrete “index-wise” constraints in (3.6) and (3.7) imply that the continuous constraint holds, but the converse does not necessarily hold true.

Expanding both the polynomial and DNN approximations leads to a family of orthogonality constraints, highlighted in Table 3.1. Intuitively, constraint C_A is the weakest of our constraints, only enforcing that the products of the DNN and polynomial in the PANN be zero. The stronger constraints C_B , C_C , and C_D ensure that during any training epoch, for a given DNN $\mathcal{N}(x)$, we find weights b_k and functions $\phi_k(x)$ in the polynomial \mathcal{P} such that their projection onto $\mathcal{N}(x)$ is zero. In a similar vein, the constraints C_E , C_F , and C_G ensure that for a given polynomial $\mathcal{P}(x)$, we determine weights a_j and functions $\psi_j(x)$ in the neural net \mathcal{N} such that their projection onto $\mathcal{P}(x)$ is zero. Constraint C_H imposes the strictest orthogonality, ensuring a higher level of independence between all pairs of polynomial and DNN bases but

TABLE 3.1
A family of orthogonality constraints

Method	Objective
C_A	$\mathcal{N}(x)\mathcal{P}(x)$
C_B	$\mathcal{N}(x)b_k\phi_k(x)$ for all $k = 1, \dots, m$
C_C	$\mathcal{N}(x)\phi_k(x)$ for all $k = 1, \dots, m$
C_D	$\mathcal{N}(x)b_k$ for all $k = 1, \dots, m$
C_E	$\mathcal{P}(x)a_j\psi_j(x)$ for all $j = 1, \dots, w$
C_F	$\mathcal{P}(x)\psi_j(x)$ for all $j = 1, \dots, w$
C_G	$\mathcal{P}(x)a_j$ for all $j = 1, \dots, w$
C_H	$a_j\psi_j(x)b_k\phi_k(x)$ for all $k = 1, \dots, m$ and $j = 1, \dots, w$

at a greater computational cost. Constraints C_B and C_D also help regularize polynomial basis coefficients, which is beneficial when the polynomial bases contain excessive terms. These constraints, coupled with our basis truncation routine, address the curse of dimensionality by eliminating unneeded bases during training.

In our experiments, however, we found that stringently enforcing any of these constraints often results in difficulties in training PANNs and is also computationally expensive in high-dimensional settings. Therefore, in this work, this discrete orthogonality is “weakly” enforced through an additional regularization loss term optimized during gradient descent in conjunction with the error norm as,

$$(3.8) \quad \min_{\theta} \|K\Psi a + K\Phi b - Kf\|_2^2 + \lambda_r \|\theta\|_1 + \lambda_c \|C\|_F,$$

where C is some constraint listed in [Table 3.1](#), λ_c is a Lagrange multiplier modulating the strength in which the orthogonality constraint is enforced, and $\|\cdot\|_F$ is the Frobenius norm. We empirically evaluate and compare each constraint listed in [Table 3.1](#) on both real and synthetic, high-dimensional problems in [section 4](#) and show that solution accuracy is generally improved through their use with minor computational expense.

3.3. Algorithm. We now outline key implementation details of the PANN optimization procedures that enhance efficiency in dynamic back-propagation frameworks like PyTorch.

Poisson Example. To illustrate our algorithmic details, we consider the Poisson equation defined by $f = \Delta u$. The following loss function describes the forward pass of our algorithm:

$$(3.9) \quad \mathcal{L} = \sum_{i=0}^{N_b} \left\| u(x_i) - \sum_{j=1}^w a_j \psi_j(x; \theta^h) - \sum_{k=1}^m b_k \phi_k(x_i) \right\|_2^2 + \sum_{i=0}^{N_r} \left\| f(x_i) - \Delta \sum_{j=1}^w a_j \psi_j(x; \theta^h) - \Delta \sum_{k=1}^m b_k \phi_k(x_i) \right\|_2^2,$$

where a_j and ψ_j represent the neural network basis coefficients and functions, respectively, while b_k and ϕ_k denote the polynomial basis coefficients and functions, as detailed in [\(3.1\)](#).

Precomputation of Polynomial Bases. The central idea is to exploit the fact that polynomial basis functions ϕ_k for $k = 1, \dots, m$ remain constant throughout training, although the coefficients b_k may change. We enhance efficiency by precomputing and

storing evaluations of these polynomial basis functions and their derivatives at the training points. Specifically, prior to the onset of training, given the set of collocation points $\mathbf{x}^r \in \mathbb{R}^{N^r \times d}$ and boundary points $\mathbf{x}^b \in \mathbb{R}^{N^b \times d}$, we store the polynomial bases evaluations $\Phi^r \in \mathbb{R}^{N^r \times m}$ and $\Phi^b \in \mathbb{R}^{N^b \times m}$. We additionally compute and store the PDE-specific derivatives, which for the Poisson problem include $\Delta\Phi^r \in \mathbb{R}^{N^r \times m}$ and $\Delta\Phi^b \in \mathbb{R}^{N^b \times m}$. Each Legendre bases evaluation uses the classical three-term recurrence relation for Legendre polynomials [25]. For classical regression problems that do not incorporate network derivatives in the forward pass, this precomputation routine is consistent across various problems. However, in the context of PDE approximation, the precomputation of polynomial bases is tailored to specific problems.

Custom Automatic Differentiation. We have developed custom forward and backward passes for the PANN architectures, specifically designed to utilize the precomputed basis functions and their derivatives. The pseudocode for these custom routines, tailored for the Poisson problem as discussed, is presented in [Algorithms 3.1](#) and [3.2](#). In these algorithms, we identify the specific gradient computations to be handled by the automatic differentiation system in blue. Specifically, line 6 of [Algorithm 3.1](#) and lines 6, 7, 8, and 10 of [Algorithm 3.2](#). All other gradients are precomputed, thereby streamlining the computational process and enhancing the efficiency of the training phase.

Algorithm 3.1 Custom Forward Pass for PANN Optimization

```

1: function FORWARDPASS( $\mathbf{x}^r, \mathbf{x}^b, \Phi^r, \Phi^b, \Delta\Phi^r, \Delta\Phi^b$ )
2:    $\mathcal{L} \leftarrow 0$  ▷ Initialize the loss function
3:    $\mathbf{u}_{\text{nn}}^b \leftarrow \Psi^b(\mathbf{x}^b; \theta^h)\mathbf{a}$  ▷ NN outputs for all boundary points
4:    $\mathbf{u}_{\text{poly}}^b \leftarrow \Phi^b\mathbf{b}$  ▷ Polynomial outputs for all boundary points
5:    $\mathcal{L} \leftarrow \mathcal{L} + \|\mathbf{u}^b - (\mathbf{u}_{\text{nn}}^b + \mathbf{u}_{\text{poly}}^b)\|_2^2$  ▷ Squared norm for boundary points
6:    $\mathbf{f}_{\text{nn}}^r \leftarrow \Delta\Psi^r(\mathbf{x}^r; \theta^h)\mathbf{a}$  ▷ NN PDE approximations for all collocation points
7:    $\mathbf{f}_{\text{poly}}^r \leftarrow \Delta\Phi^r\mathbf{b}$  ▷ Polynomial PDE approximations for all collocation points
8:    $\mathcal{L} \leftarrow \mathcal{L} + \|\mathbf{f}^r - (\mathbf{f}_{\text{nn}}^r + \mathbf{f}_{\text{poly}}^r)\|_2^2$  ▷ Squared norm for collocation points
9:   return  $\mathcal{L}$  ▷ Return the computed loss
10: end function

```

Algorithm 3.2 Custom Backward Pass for PANN Optimization

```

1: function BACKWARDPASS( $\mathbf{x}^r, \mathbf{x}^b, \Phi^r, \Phi^b, \Delta\Phi^r, \Delta\Phi^b$ )
2:   Initialize  $\nabla\theta^h, \nabla\mathbf{a}, \nabla\mathbf{b}$  to zeros
3:    $\mathbf{g}^b \leftarrow 2(\mathbf{u}^b - (\Psi^b(\mathbf{x}^b; \theta^h)\mathbf{a} + \Phi^b\mathbf{b}))$  ▷ Gradients w.r.t. boundary outputs
4:    $\nabla\mathbf{a} \leftarrow -(\Psi^b(\mathbf{x}^b; \theta^h))^T\mathbf{g}^b$  ▷ Gradient w.r.t. NN coefficients at boundaries
5:    $\nabla\mathbf{b} \leftarrow -(\Phi^b)^T\mathbf{g}^b$  ▷ Gradient w.r.t. polynomial coefficients at boundaries
6:   Update gradients of  $\theta^h$  based on  $\mathbf{g}^b$  and derivative computations for  $\Psi^b$ 
7:    $\mathbf{g}^r \leftarrow 2(\mathbf{f}^r - (\Delta\Psi^r(\mathbf{x}^r; \theta^h)\mathbf{a} + \Delta\Phi^r\mathbf{b}))$  ▷ Gradients w.r.t. collocation PDE residuals
8:    $\nabla\mathbf{a} \leftarrow \nabla\mathbf{a} - (\Delta\Psi^r(\mathbf{x}^r; \theta^h))^T\mathbf{g}^r$  ▷ NN coefficients gradients at collocations
9:    $\nabla\mathbf{b} \leftarrow \nabla\mathbf{b} - (\Delta\Phi^r)^T\mathbf{g}^r$  ▷ Polynomial coefficients gradients at collocations
10:  Update gradients of  $\theta^h$  based on  $\mathbf{g}^r$  and derivative computations for  $\Delta\Psi^r$ 
11:  return  $\nabla\theta^h, \nabla\mathbf{a}, \nabla\mathbf{b}$  ▷ Return gradients for updating parameters
12: end function

```

Basis Truncation. We apply L_1 regularization to both the DNN coefficients a_j and the coefficients of the polynomial layer bases b_k , truncating any coefficients that fall below a specified threshold t . Specifically, we set the coefficients defined by $\{a_j, b_k \mid a_j < t \text{ for } j = 1, \dots, w \text{ and } b_k < t \text{ for } k = 1, \dots, m\}$ to zero, along with their corresponding basis functions. This truncation strategy is critical given the po-

tentially large number of basis functions m in high-dimensional problems, as demonstrated in [Figure 2.1](#). These precomputations are particularly beneficial in applications such as solving PDEs, where the n th derivatives of each polynomial basis are required. By pre-computing and storing these derivatives, we significantly reduce the computational load during each training iteration, thus enhancing overall efficiency.

Computational Complexity. Focusing specifically on Legendre polynomials, the recursive evaluation of the m th Legendre polynomial (or its derivative) at a single point has a computational complexity of $O(m)$, and $O(Nm)$ for N points. In [\(3.9\)](#), if we set $N = \max(N_r, N_b)$, the additional computational cost of polynomial augmentation in both the forward and backward passes is $O(Nm)$. While the precomputation strategy’s asymptotic computational cost remains $O(Nm)$, it reduces the constant factors within $O(Nm)$ by almost half, resulting in significant practical improvements. Additionally, implementing the parallel computing methods described in [\[8\]](#) for $O(1)$ computation of Legendre polynomials could improve complexity to just $O(N)$. The basis coefficient truncation strategy defined above further reduces the number of active bases while enhancing computational efficiency during training.

PyTorch C++. We remark that our code was written in C++ using the PyTorch C++ library [\[58\]](#), which offers several compelling benefits over other pure Python methods for developing machine learning and deep learning applications. PyTorch C++ inherits many of the strengths of the Python version of PyTorch, such as its dynamic computational graph, while bypassing Python’s often slow interpretation. The PyTorch C++ library, as of this writing, is an underutilized tool in the research community despite being significantly faster than its Python counterpart. Therefore, all C++ source code for our methods (and baselines) is open-source and publicly available¹, facilitating extensions and further investigation for the research community.

4. Numerical Experiments. In this section, we explore the effectiveness of PANNs through a series of detailed numerical experiments that compare them to a range of established methods. Our evaluation includes comparisons with deep neural networks (DNNs) utilizing Tanh, ReLU, and RePU activations. Additionally, we examine the performance of a standalone polynomial layer (PL) trained via gradient descent, which serves as a simplified version of PANN, as well as polynomial least squares using Legendre polynomials (L^2). [Table 4.1](#) lists the specific advantages and disadvantages of each baseline method. Furthermore, the appendix includes the benchmark results against various conventional regression models.

We assess model performance using the relative ℓ_2 error, defined as $\frac{\|y - \hat{y}\|_2}{\|y\|_2} = \frac{\sqrt{\sum_{i=1}^n (y_i - \hat{y}_i)^2}}{\sqrt{\sum_{i=1}^n y_i^2}}$, where \hat{y} is the predicted solution and y is the true solution. This metric is beneficial in scientific computing applications where the magnitude of the data plays an important role, helping to understand the error’s significance relative to the data’s scale.

All tests involving PANN and the polynomial layer (PL) incorporate preconditioning and basis truncation. All L^2 projection tests also utilize preconditioning. We perform five random trials for each synthetic dataset and report the mean ℓ_2 errors and the standard deviations. We also include wall-clock training times for each experiment, expressed in seconds. We conduct a four-fold cross-validation for real-world data applications and present the mean ℓ_2 errors and standard deviations. The appen-

¹pending publication

TABLE 4.1

Comparative analysis of various function approximation methods, delineating their objectives, advantages, and disadvantages

Method	Objective	Pros	Cons
a. Neural Network (Tanh Activation)	Minimize mean squared error (MSE).	Efficient training (less risk of vanishing gradients); handles high-dimensional data well.	High computational load with larger networks or dimensions.
b. Neural Networks (ReLU Activation) [52]	Minimize mean squared error (MSE).	Efficient training; handles complex, high-dimensional problems.	Risk of vanishing/exploding gradients. Requires careful initialization and optimization methods.
c. Neural Networks (RePU Activation) [42, 66]	Minimize mean squared error (MSE).	Excels at approximating smooth functions.	Works poorly for non-smooth functions.
d. L^2 Projection with orthogonal Polynomials	Solve $\langle f - \hat{f}, v \rangle = 0$ for all v in V , using quadrature.	Provides high accuracy for smooth functions; efficient for regular domains and low-dimensional problems.	Not suited for non-smooth functions [18, 22]; requires exponentially more points with increasing dimensions.
e. Single Polynomial Layer (PL) with Legendre Basis	Minimize mean squared error (MSE).	Adaptable to a broad range of problems; no structured point requirements; optimized through gradient descent.	Possibly less precise than method ‘d’; may struggle with very high-dimensional data.

dix includes detailed information on each experiment’s implementation and training specifics.

4.1. Legendre Polynomial Approximation. In assessing the effectiveness of PANNs, it is critical to accurately recover polynomial functions, especially when evaluating whether the additional DNN component of the architecture impacts the solution accuracy of the polynomial layer. High-order Legendre polynomials, such as the tenth order, are highly oscillatory—a condition known to challenge DNNs as previously mentioned. However, polynomial methods equipped with sufficient bases can recover polynomial solutions exactly. To illustrate this, we introduce a test scenario where the ground truth is a two-dimensional, tenth-order Legendre polynomial, defined as:

$$(4.1) \quad u(x) = u(x, y) = P_{10}(x)P_{10}(y),$$

such that P_{10} is the 10th Legendre polynomial given by

$$P_{10}(z) = \frac{46189}{256}z^{10} - \frac{109395}{256}z^8 + \frac{90090}{256}z^6 - \frac{30030}{256}z^4 + \frac{3465}{256}z^2 - \frac{63}{256}.$$

In this test, effective basis truncation within PANN is essential. If the polynomial bases adequately span the true solution, then truncating 100% of the DNN bases and 99.7% of the polynomial bases should leave just one active polynomial base, expected to be $\phi = P_{10}(x)P_{10}(y)$ with a coefficient $b = 1$. This experiment demonstrates

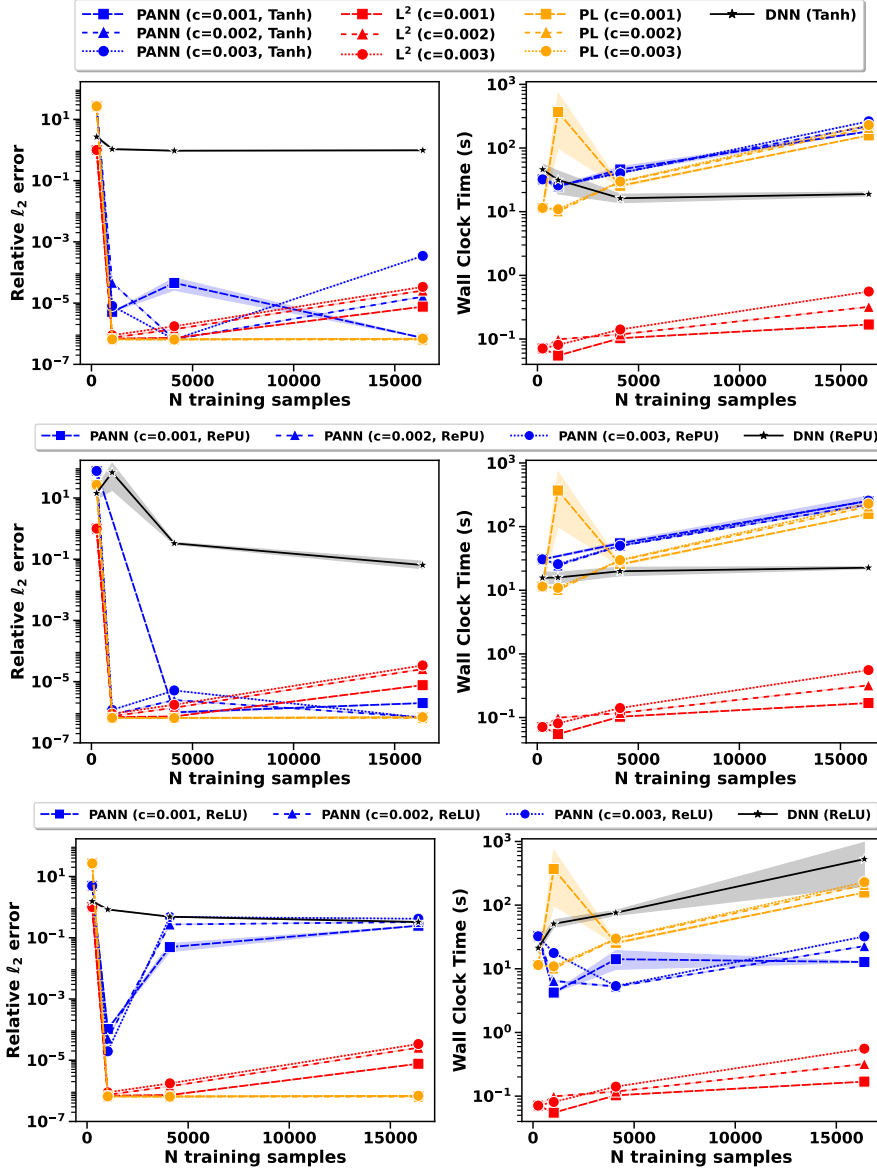


FIG. 4.1. (Left) Relative ℓ_2 errors and (right) wall clock time in seconds for different network types using the Tanh (top), RePU (middle), and ReLU (bottom) activation function. PL and L^2 projection results are repeated in each figure for easy comparison.

PANNs’ improved ability to reproduce polynomials, such that with the proper orthogonality constraints and our basis truncation technique, PANNs can achieve polynomial solutions with near-machine precision. This confirms that the DNN component does not compromise the approximation accuracy.

We evaluate the relative ℓ_2 errors in the solution as a function of the number of training points N , with N values set at 256, 1024, 4096 and 16384. We sample points using the efficient Poisson sampling technique, as outlined in [64] in the domain $[-1, 1]$. For experiments involving PANN, the single polynomial layer (PL), and L^2

TABLE 4.2

Suite of total degrees ℓ and corresponding count of polynomial terms m in polynomial bases used in the exact solution recovery ($u(x, y) = P_{10}(x)P_{10}(y)$) example. The total degree is set to $\ell = 2(\lceil cN \rceil + 8)$.

	c = 0.001				c = 0.002				c = 0.003			
N	256	1024	4096	16384	256	1024	4096	16384	256	1024	4096	16384
ℓ	18	20	26	50	18	22	34	82	18	24	42	116
m	190	231	378	1326	190	276	630	3486	190	325	946	6903

projection, we varied the total degree of Legendre bases by the number of training samples. Specifically, for constants $c = 0.001, 0.002$, and 0.003 , we determine the total degree is $\ell = 2(\lceil cN \rceil + b)$ where $b = 8$. Detailed configurations of total degrees ℓ and the corresponding widths of the polynomial layer m are documented in Table 4.2. Note that the total degree of the true solution u is $\ell = 20$, meaning for $\ell < 20$, the polynomial bases do not contain the true solution which occurs for $N = 256/c = 0.001, 0.002$. For both the standard DNNs and the DNN component of PANN, we use three hidden layers each with 100 neurons, and compare RePU, ReLU, and Tanh activation functions. We also apply orthogonality constraint C_E in PANN. Table A.1 in the Appendix presents the error results compared to various popular regression models.

The results presented in Figure 4.1 confirm that PANNs can either exactly or almost exactly recover polynomial functions, particularly when using the Tanh and RePU activation functions. This success likely stems from the smooth nature of both activations and the true solutions. The discrepancy in the performance of PANNs when employing ReLU, as opposed to Tanh or RePU, is likely attributed to the employed orthogonality constraint C_E . Given that the true solution necessitates a negligible contribution from the DNN component and solely a single polynomial basis, stringent coefficient regularization (and thus truncation) is required. Constraint C_E seeks a DNN weight a_j and a corresponding basis function ψ_j that collectively project to zero onto the polynomial output. This particular problem hints that the emphasis on DNN basis function optimization might over-complicate the learning by trying to match non-smooth DNN bases with a smooth target, thereby compromising the optimization of the polynomial layer.

In general, when equipped with smooth activation functions and an adequate number of polynomial bases (that is, when $\ell > 20$), PANNs can almost perfectly replicate the true solutions. Notably, with Tanh and RePU activations, PANNs outperforms the accuracy of L^2 projection, with a larger number of training points and polynomial total degrees. The higher error in the L^2 projection method is likely due to numerical issues related to the large number of unnecessary polynomial bases. Implementing basis truncation could reduce errors to negligible levels, as seen in the PL results. Remarkably, optimizing coefficients using gradient descent in the PL method achieves comparable accuracy to traditional L^2 projection methods despite using Poisson-distributed points instead of quadrature points. This suggests that gradient descent optimization can deliver solutions as precise as projection methods without requiring structured training points. Consequently, this opens up the possibility for more flexible extensions to irregular domains and higher-dimensional problems, though it may come at a more significant computational cost. These results indicate that jointly optimizing both the DNN and polynomial coefficients in PANNs through gradient descent is a valid approach. The spike in training time for

the PL method when $N = 1024$ and $c = 0.001$ is likely due to resource contentions within the system. The standard DNNs exhibit relatively high errors irrespective of the activation function, underscoring the inherent challenges they face in approximating highly-oscillatory solutions.

The subsequent tests will extend this comparison to include each orthogonality constraint alongside various activation functions. We anticipate demonstrating that PANNs with ReLU activations, when paired with orthogonality constraints that de-emphasize DNN basis optimization (such as C_B or C_G), can also achieve nearly exact recovery of the ground truth solutions. This would further validate the flexibility and robustness of PANNs in accurately approximating polynomial functions.

4.1.1. Orthogonality Constraint Comparisons. We examined the impact of each discrete orthogonality constraint described in Table 3.1 on solution accuracy and computational overhead against baselines using no constraints (labeled ‘None’) and L_1 coefficient regularization. Our experiments use $N = 4096$ training points, a polynomial layer with $\ell = 26$, and width $m = 378$. Introducing orthogonality constraints generally improves solution quality, as demonstrated in Figure 4.2. Specifically, the Tanh activation yielded lower relative ℓ_2 errors with constraints C_E and C_H , while ReLU is more accurate with constraints C_A and C_B . The relative performance between ReLU and Tanh using other constraints was marginal. ReLU activation paired with constraint C_F did not converge, potentially due to ReLU’s properties and the constraint overemphasizing the independence between the polynomial solution and DNN bases that are not well-suited to the target function. Conversely, constraint C_G achieved the highest accuracy across all activations and reduced training times, suggesting that judicious constraint selection can expedite convergence. Moreover, constraint C_G exactly recovers the intended basis functions through truncation; Table 4.3 shows that 100% of the DNN bases were truncated, and 99.7% of the polynomial bases were truncated. This suggests that the chosen constraints contribute to models that closely resemble the intended solution, outperforming both L_1 normalization and scenarios devoid of constraints.

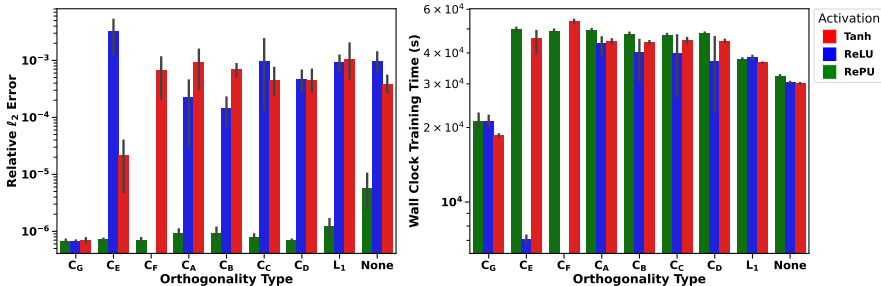


FIG. 4.2. (Left) Barplot showing the relative ℓ_2 errors for each orthogonality constraint and activation, compared to using no constraint (None) and using standard L_1 regularization. The (right) barplot shows the wall clock training times of the associated method in seconds.

While the RePU activation performs best across all constraint variants, it often incurs slightly higher training times, and interestingly, does not recover the same bases sets as the true solution. This indicates that the polynomial activations in the DNN and the polynomial layer both model different portions of the target, and the improved error results of all constraint variations (over L_1 normalization and no constraints) indicates that the orthogonality aids optimization even in settings where

TABLE 4.3

The percentage of truncated NN and polynomial bases coefficients in the form %NN/%PL. Bold values represent methods who truncated the expected number of bases.

Act.	C_G	C_E	C_F	C_A	C_B	C_C
Tanh	100/99.7	97.6/ 99.7	52.4/ 99.7	51.8/ 99.7	52.8/99.6	50.0/ 99.7
ReLU	100/99.7	100 /73.8	—	59.0/ 99.7	58.6/ 99.7	60.6/92.9
RePU	100/99.7	53.4/ 99.7	46.8/ 99.7	51.0/ 99.7	53.2/ 99.7	50.0/ 99.7

Act.	C_D	L_1	None
Tanh	52.0/ 99.7	47.8/99.4	47.4/97.1
ReLU	71.8/ 99.7	49.4/99.6	50.6/84.2
RePU	53.8/ 99.7	46.2/ 99.7	48.2/99.6

the DNN and polynomial expressivity are comparable.

4.2. Approximating Non-Smooth Functions. The previous experiment demonstrated that PANNs can recover polynomial solutions as effectively as traditional methods, known for their robust handling of smooth functions. It also confirmed that the DNN component within PANNs does not compromise polynomial solution recovery. Given the known flexibility of DNNs to handle complex and nonlinear functions, this test aims to explore the converse of our previous findings. Specifically, we want to ensure that the polynomial layer in PANNs, typically less adept at managing non-smooth functions, does not hinder the DNN portion’s ability to effectively approximate these types of functions. Furthermore, we seek to show that the integrated approach of PANNs, in fact, reduces approximation errors compared to standard DNNs alone. Therefore, we evaluate the performance of PANNs against baseline methods by approximating a manufactured two-dimensional non-smooth function:

$$(4.2) \quad u(x, y) = x^2 \sin(1/y),$$

which belongs to the function space $C^1(\mathbb{R}^2)$. This function poses a significant challenge due to its discontinuity at $y = 0$, testing the capability of PANNs to handle complexities beyond those that traditional polynomial and neural network methods typically address. We use a consistent experimental setup to [subsection 4.1](#), but varied the total degree of the Legendre bases by $\ell = \lceil cN \rceil + b$ for $b = 8$. All total degree and polynomial layer width configurations are documented in [Table 4.4](#).

TABLE 4.4

Suite of total degrees ℓ and corresponding count of polynomial terms m in polynomial bases used in the non-smooth function ($u(x, y) = x^2 \sin(1/y)$) approximation example.

	$c = 0.001$				$c = 0.002$				$c = 0.003$			
N	256	1024	4096	16384	256	1024	4096	16384	256	1024	4096	16384
ℓ	9	10	13	25	9	11	17	41	9	12	21	58
m	55	66	105	351	55	78	171	903	55	91	253	1770

As depicted in [Figure 4.3](#), the L^2 projection method struggles to find accurate solutions and obtains a minimum error of $1e-2$ when using 16,384 quadrature points and a Legendre bases with total degree set to $\ell = 58$ with $m = 1,770$ total polynomial bases. The standard DNNs also struggle to find accurate solutions when using Tanh activations, though obtains superior accuracy over L^2 projection for training

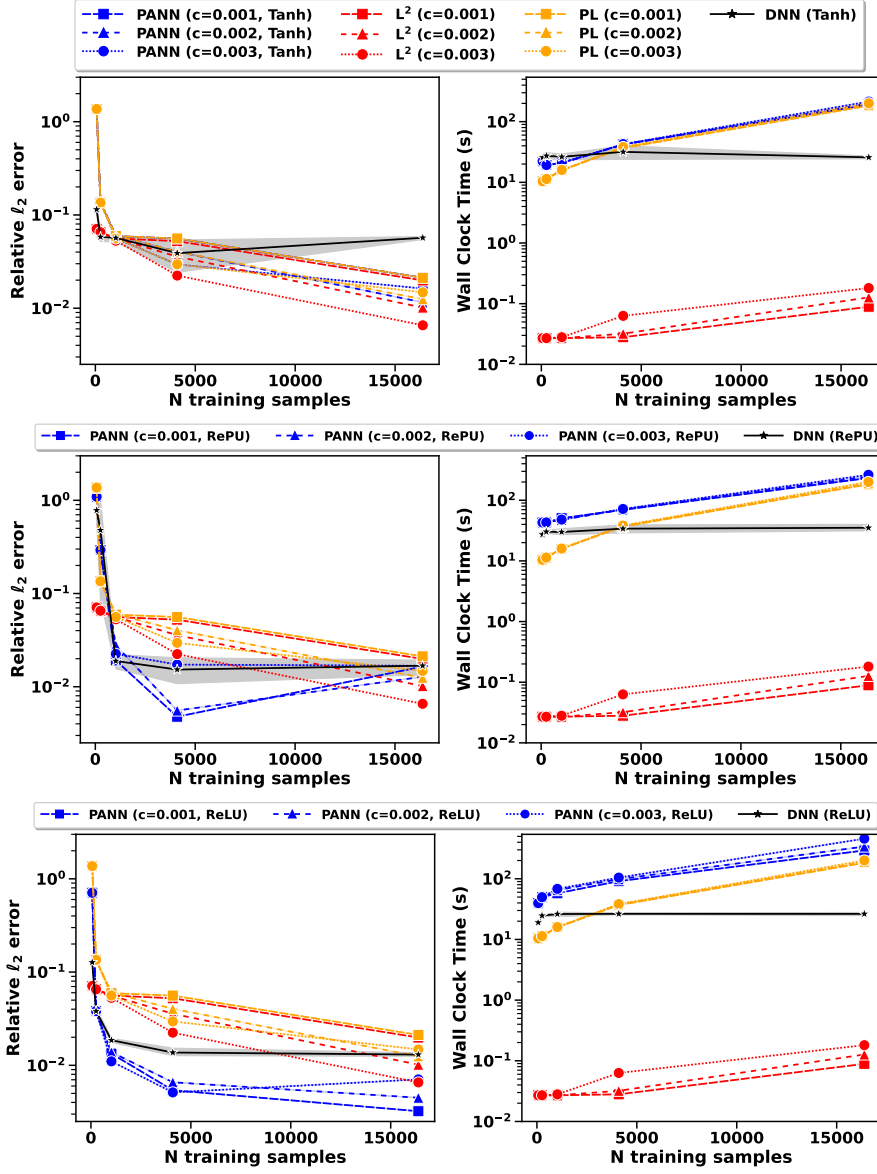


FIG. 4.3. (Left) Relative l_2 errors and (right) wall clock time in seconds for different network types using the Tanh (top), RePU (middle), and ReLU (bottom) activation function. PL and L^2 projection results are repeated in each figure for easy comparison.

set/quadrature point sizes under 16,384 when using ReLU and RePU. PANN demonstrates superior predictive performance, especially notable when using ReLU activation across moderate and large training set sizes. Even under conditions with smaller sets, PANN maintains improved accuracy over the L^2 projection method, suggesting a robust ability to approximate C^1 functions with ReLU more effectively than other tested methods, including those listed in Table A.2. Despite the computational overhead, polynomial layers (PLs) show less accuracy for the same computational cost as PANN, and standard DNNs fail to match the performance improvements observed

with PANN, regardless of increases in training data volume. These findings advocate for the integrated approach of PANN, where neither PLs nor DNNs capture the solution adequately when used as distinct approaches.

4.3. High-Dimensional and Noisy Target Functions. In the previous two sections, we have demonstrated the validity of PANNs in low-dimensional problems. This section shows results for a high-dimensional synthetic problem and a high-dimensional real-world problem. We show that with our implementation and optimization techniques, PANNs can achieve better accuracy in these cases compared to standard regression and DNN methods, further validating our enriched DNN approach.

4.3.1. High-Dimensional Synthetic Example. In this section, we explore the capabilities of PANN and DNNs using Tanh, ReLU, and RePU activations, and polynomial layers (PLs), in approximating high-dimensional synthetic functions of the form:

$$(4.3) \quad u(x) = 5\pi^2 \sin(2\pi x_0) \prod_{i=1}^d \sin(\pi x_i),$$

across dimensions ranging from two to six. We set the number of training points to $N = 1536, 3072, 6144, 12288, \text{ and } 26576$, corresponding to each dimension. For the PANNs, we consistently applied preconditioning and the orthogonality constraint C_E , with a fixed total degree for the polynomial bases set at $\ell = 8$. This corresponds to polynomial layers of width $m = 45, 165, 495, 1287, 3003$ for dimensions two through six respectively. For both the standard DNNs and the DNN component of PANN, we use three hidden layers each with 100 neurons. Additionally, [Table A.3](#) in the Appendix presents the error results compared to various popular regression models. Due to the prohibitive computational complexity of traditional L^2 projection methods in high dimensions—mainly from the need to generate extensive quadrature point sets—we excluded these from our comparisons. Our focus remains on straightforward, scalable methodologies that require minimal preprocessing and are suitable for various high-dimensional contexts.

As illustrated in [Figure 4.4](#), PANN with Tanh activation consistently delivered the lowest relative ℓ_2 errors across all tested dimensions, albeit with a modest increase in both error and computational time as the problem dimension increased. In contrast, PANN equipped with RePU activation demonstrated higher errors and more significant increases in computational time, making it less suited for scaling to larger dimensions.

4.3.2. Noisy Real-World Example. We next evaluate PANNs’ effectiveness in predicting housing prices from the California housing dataset [\[56\]](#), which includes features such as the number of bedrooms, occupancy rates, and median house values across 20,640 instances with eight total features. We compare each orthogonality constraint described in [Table 3.1](#), assessing their performance under preconditioned and non-preconditioned settings. Additionally, we compare PANNs with DNNs employing ReLU and Tanh activation functions, excluding RePU due to its failure to converge in preliminary tests. This convergence issue likely arises from the inability of the polynomial components in both DNNs and PANNs to accurately model sharp transitions in the data. For both the standard DNNs and the DNN component of PANN, we use three hidden layers each with 100 neurons. Each feature was normalized to the range $[-1, 1]$, and our results are derived from a four-fold cross-validation

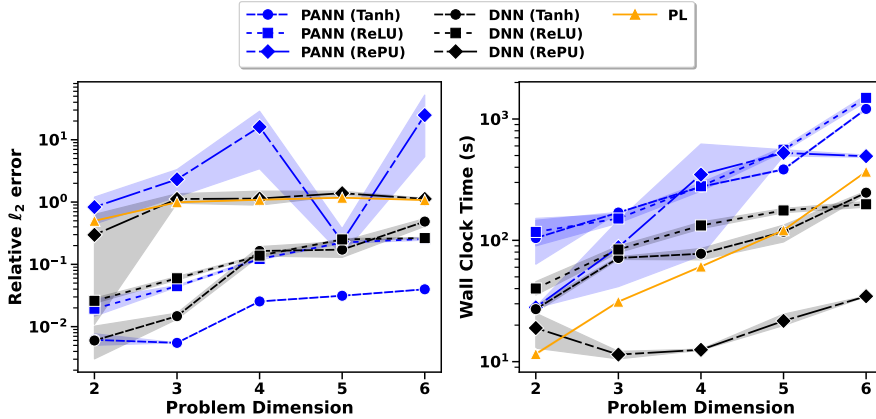


FIG. 4.4. (Left) Relative ℓ_2 error by problem dimension and (right) wall clock time in seconds by problem dimension for DNNs and PANNs using Tanh, ReLU, and RePU activation functions compared to the polynomial layer (PL).

process, with each fold comprising 15,480 training samples and 5,152 test samples. We provide comparisons to popular regression models in Table A.4 in the Appendix.

The results, displayed in Table 4.5, show that preconditioning generally improves the performance of PANNs by reducing the relative ℓ_2 error. Notably, this improvement is generally consistent across both ReLU and Tanh activations, indicating that the benefits of preconditioning transcend the choice of the activation function. Interestingly, using preconditioning with constraint C_H increased errors for both activations compared to using no preconditioning. Among the tested configurations, PANN employing the C_E constraint with ReLU activation and preconditioning demonstrated the most effective error reduction. This suggests that the smaller polynomial bases efficiently capture general data trends, while the DNN components, constrained by C_E , effectively model the residual complexities, possibly including noise and non-linear transitions. PANN generally surpasses traditional DNNs in terms of error when comparing similar activations, likely due to its enhanced capacity for modeling intricate relationships within the data. Several PANN configurations without preconditioning achieve greater efficiency and superior performance than standard DNNs, underscoring their viability in scenarios where both computational efficiency and predictive accuracy are desired. Moreover, the analysis of polynomial layer (PL) variants indicates a notably higher error and longer training durations, suggesting these configurations might be less appropriate for this dataset.

4.4. Physics-Informed PANNs. In this section, we exhibit the simplicity for which our methods extend to other settings such as PDE solution approximation. As mentioned previously, our goal is to augment the Physics-Informed Neural Network (PINN) $\mathcal{N}(x)$, with the polynomial bases $\mathcal{P}(x)$ resulting in a new approximation method which we call **Physics-Informed Polynomial-Augmented Neural Networks (PI-PANNs)** $u_\theta(x)$ as in (3.1). We present results comparing PI-PANNs to a variety of standard PINNs on the 2D Poisson and 2D (steady-state) Allen-Cahn problems.

The linear Poisson equation given by $\Delta u(x) = f(x)$ for $x \in \Omega$ and $u(x) = g(x)$ for $x \in \partial\Omega$. The steady-state Allen-Cahn equation—a non-linear elliptic problem—is given by $\Delta u + u(u^2 - 1) = f(x)$ for $x \in \Omega$ and $u(x) = g(x)$ for $x \in \partial\Omega$, where

TABLE 4.5

Relative ℓ_2 errors and training times for PANNs for each constraint with and without preconditioning, DNNs and polynomial layers (PLs) under various settings.

Network	Act	Precond	Relative ℓ_2 Error	Wall-clock Time (s)
PANN- C_A	ReLU	False	0.2256 ± 0.002	165.5032 ± 272.636
		True	0.2126 ± 0.002	1075.4628 ± 1861.269
	Tanh	False	0.2249 ± 0.019	126.4284 ± 187.072
		True	0.2202 ± 0.003	1546.4286 ± 2925.925
PANN- C_B	ReLU	False	0.2164 ± 0.006	152.2832 ± 242.852
		True	0.2139 ± 0.003	1084.5254 ± 1884.902
	Tanh	False	0.2200 ± 0.009	114.0070 ± 196.716
		True	0.2209 ± 0.003	1063.0726 ± 1855.294
PANN- C_C	ReLU	False	0.2213 ± 0.004	151.7962 ± 246.296
		True	0.2124 ± 0.002	238.2235 ± 4.155
	Tanh	False	0.2639 ± 0.003	132.9082 ± 206.539
		True	0.2239 ± 0.004	231.4235 ± 2.245
PANN- C_D	ReLU	False	0.2615 ± 0.006	160.4952 ± 266.624
		True	0.2120 ± 0.003	241.8500 ± 1.410
	Tanh	False	0.2645 ± 0.003	142.3642 ± 225.477
		True	0.2183 ± 0.004	233.1430 ± 2.675
PANN- C_E	ReLU	False	0.2122 ± 0.003	190.5806 ± 322.992
		True	0.2118 ± 0.005	242.8420 ± 3.402
	Tanh	False	0.2319 ± 0.013	128.1918 ± 190.995
		True	0.2225 ± 0.006	239.5772 ± 1.154
PANN- C_F	ReLU	False	0.2153 ± 0.002	44.3470 ± 0.790
		True	0.2123 ± 0.003	242.0880 ± 3.468
	Tanh	False	0.2291 ± 0.002	157.4134 ± 260.070
		True	0.2288 ± 0.004	233.9955 ± 4.150
PANN- C_G	ReLU	False	0.2450 ± 0.020	92.6790 ± 143.156
		True	0.2126 ± 0.001	233.2752 ± 4.080
	Tanh	False	0.2189 ± 0.004	133.5222 ± 221.093
		True	0.2214 ± 0.003	225.1395 ± 2.406
PANN- C_H	ReLU	False	0.2120 ± 0.004	360.6397 ± 465.925
		True	0.2123 ± 0.006	330.7845 ± 5.975
	Tanh	False	0.2253 ± 0.004	278.8388 ± 345.442
		True	0.2268 ± 0.001	323.1293 ± 3.634
DNN	ReLU	False	0.2132 ± 0.003	317.4062 ± 23.341
	Tanh	False	0.2157 ± 0.003	291.6603 ± 7.065
PL ($l = 2$)	—	True	0.2538 ± 0.003	206.6065 ± 2.536
PL ($l = 4$)	—	True	0.2654 ± 0.029	258.8638 ± 7.315
PL ($l = 6$)	—	True	0.8696 ± 0.170	331.7095 ± 13.724

$\Omega = [-1, 1]^2$. f and g are given in both problems where the goal is to recover u . For testing the 2D Poisson equation, similar to [subsection 4.1](#), we manufacture a solution where the ground truth is $u(x, y) = P_{10}(x)P_{10}(y)$, the tenth Legendre polynomial, from which we derive f and g . Then, to investigate how our method performs on non-linear PDEs, we set the true solution u in the 2D Allen-Cahn equation to be

$$u(x, y) = x^3y^3 + 5x \cos(2\pi x) \cos(2\pi y).$$

We tested each using three architecture settings; three hidden layers with 50 nodes, three hidden layers with 100 nodes, and five hidden layers with 50 nodes. We trained each model using 64, 256, 1024 and 4096 collocation points, and 400 boundary points on both randomly sampled and equispaced points on the interval $[-1, 1]^2$. For PI-PANNs, we varied the polynomial complexity by the number of training points based on two different functions of the N , and we apply orthogonality constraint C_E . Specifically, for $c = 0.003, 0.004$, the total degree of the Legendre bases is $\ell = 2(\lceil cN \rceil + 8)$; detailed configurations of total degrees ℓ and the corresponding widths of the polynomial layer m are documented in [Table 4.6](#).

TABLE 4.6

Suite of total degrees ℓ and corresponding count of polynomial terms m in polynomial bases used in the PDE approximation examples.

	$c = 0.003$				$c = 0.004$			
N	64	256	1024	4096	64	256	1024	4096
ℓ	18	18	24	42	18	20	26	50
m	190	190	325	946	190	231	378	1326

[Figures 4.5](#) and [4.6](#) compare the relative errors for the Poisson and Allen-Cahn problems, respectively, as achieved by different methods. The standard PINNs consistently fail to train under all settings, with relative errors not dropping below 0.53 for the Poisson problem and 0.34 for the Allen-Cahn problem as shown in [Figure 4.5](#). This lack of training effectiveness underscores the standard PINNs’ sensitivity to hyperparameters and training routines, in contrast to the more robust PI-PANNs. In contrast, PI-PANNs achieve relative errors around $7e - 05$ in both the Allen-Cahn and Poisson examples using 4046 collocation points in the Allen-Cahn example and only 256 in the Poisson example. Interestingly, the results also suggest that PI-PANNs may have a slight preference for equispaced training points but provide consistently good performance across both point sampling techniques, albeit at an increased computational training time.

5. Conclusion and Future Work. This paper proposes an effective and applicable method of augmenting neural networks with a trainable polynomial layer. Additionally, we provide a suite of novel discrete orthogonality constraints enforced through the loss function during optimization. Through a suite of numerical experiments, we show that—although simple—our methods result in higher accuracy across a broad range of test problems and apply to many domains, such as predicting solutions to PDEs. The experiments show that while our methods increase accuracy, including the polynomial preconditioning increases training times. Investigating efficient polynomial preconditioners for polynomials used in neural network architectures would be an interesting future research direction. Additionally, investigating PI-PANNs to solve space-time PDEs is left for future work.

Appendix A. Experimental Settings and Additional Numerical Results.

In this section, we detail the specific experimental settings used in each experiment, along with additional numerical results for the standard benchmark methods on the regression examples.

A.1. Experimental Details. The experiments were conducted on a GeForce RTX 3090 GPU with CUDA version 12.3, running on Ubuntu 20.04.6 LTS. We used total degree Legendre polynomial bases in all relevant models. The orthogonality

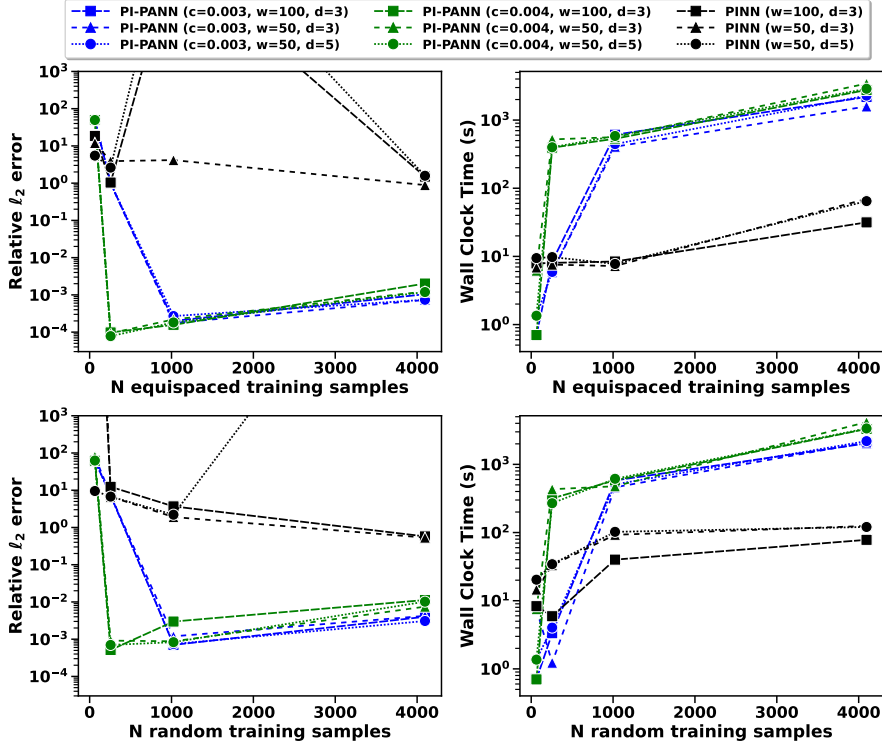


FIG. 4.5. (Left) Relative l_2 errors and (right) wall clock time in seconds solving the Poisson problem for different network types using equispaced collocation points (top), and randomly spaced collocation points (bottom). Note that the scale of the plots is intentionally maintained to ensure clarity and comparability of the results across different model configurations. Adjusting the scale to include outlier values from PINN results obscures critical differences among error values.

strength (λ_c) in PANNs is 0.001, and the basis coefficient truncation threshold is 0.0001. We optimized each model using 20,000 Adam iterations with an initial learning rate of 0.001, and 400 LBFGS iterations with an initial learning rate set to 1.0. We also employed the cosine learning rate annealing method [49]. We used Gauss-Legendre quadrature for all L^2 projection experiments.

A.2. Benchmarking Methods. We compared our results against a variety of standard regression models to benchmark performance. These models include: AdaBoost, Bagging, Bayesian Ridge, Elastic Net, Gradient boosting, Huber regression, linear SVR, MLP, Nu SVR, SVR, and kNeighbors regression. Tables A.1 to A.4 list the models whose relative errors fall below at least 0.6 for each example.

TABLE A.1

Relative l_2 errors and standard deviations of baseline methods for predicting the two-dimensional Legendre function problem in subsection 4.1

Model	64	256	1024	4096	16384
Bagging	0.916 ± 0.076	0.854 ± 0.014	0.881 ± 0.021	0.734 ± 0.020	0.433 ± 0.017
kNeighbors	1.064 ± 0.000	0.972 ± 0.000	0.876 ± 0.000	0.619 ± 0.000	0.333 ± 0.000

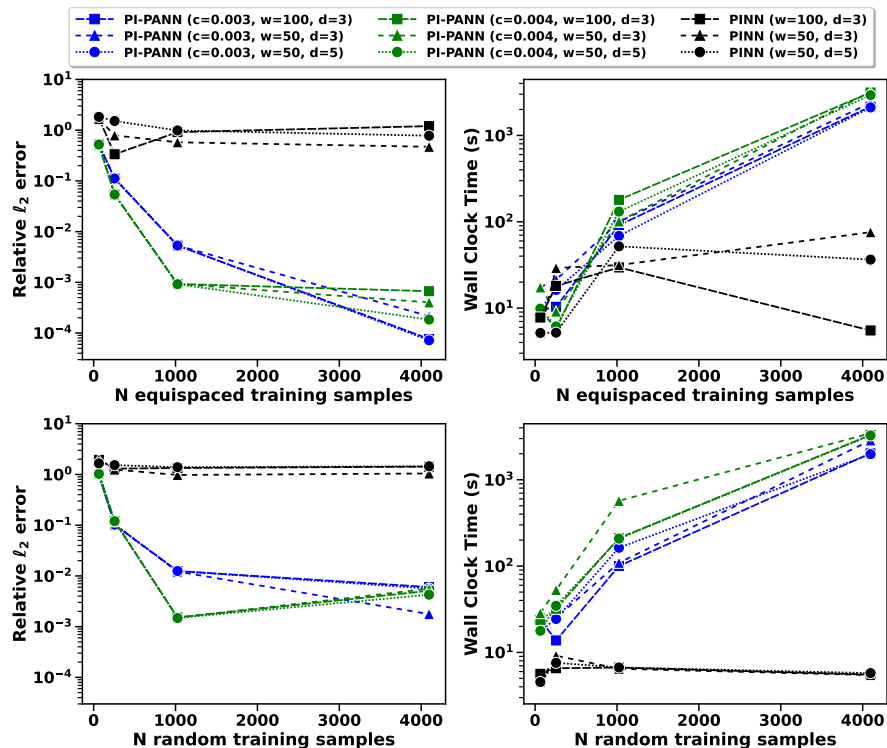


FIG. 4.6. (Left) Relative ℓ_2 errors and (right) wall clock time in seconds solving the Allen-Cahn problem for different network types using equispaced collocation points (top), and randomly spaced collocation points (bottom).

TABLE A.2

Relative ℓ_2 errors and standard deviations of baseline methods for predicting the non-smooth synthetic function in subsection 4.2.

Model	64	256	1024	4096	16384
Bagging	1.190 ± 0.220	0.462 ± 0.084	0.180 ± 0.022	0.073 ± 0.003	0.033 ± 0.006
Grad Boost	0.846 ± 0.010	0.572 ± 0.003	0.240 ± 0.001	0.184 ± 0.000	0.217 ± 0.000
MLP	0.778 ± 0.068	0.864 ± 0.191	0.528 ± 0.164	0.359 ± 0.077	0.313 ± 0.042
Nu SVR	0.418 ± 0.000	0.151 ± 0.000	0.077 ± 0.000	0.069 ± 0.000	0.060 ± 0.000
SVR	0.690 ± 0.000	0.346 ± 0.000	0.217 ± 0.000	0.190 ± 0.000	0.170 ± 0.000
kNeighbors	0.776 ± 0.000	0.189 ± 0.000	0.081 ± 0.000	0.041 ± 0.000	0.018 ± 0.000

TABLE A.3

Relative ℓ_2 errors and standard deviations of baseline methods for increasing dimensions from subsection 4.3.1.

Model	2	3	4
Bagging	0.122 ± 0.004	—	0.750 ± 0.265
Bayesian Ridge	1.005 ± 0.010	0.237 ± 0.329	—
Elastic Net	1.000 ± 0.000	0.550 ± 0.606	—
kNeighbors	0.175 ± 0.000	—	0.479 ± 0.035

Acknowledgments. We acknowledge the use of large language models (LLMs) for manuscript editing.

TABLE A.4

Relative ℓ_2 errors and standard deviations of baseline methods for the real-world dataset from subsection 4.3.2.

Model	Relative ℓ_2 Error	Model	Relative ℓ_2 Error
Adaboost	0.368 ± 0.016	MLP	0.474 ± 0.084
Bagging	0.225 ± 0.003	Nu SVR	0.263 ± 0.003
Elastic Net	0.487 ± 0.002	SVR	0.264 ± 0.003
Grad Boosting	0.225 ± 0.004	Theil Sen	0.511 ± 0.263
Huber	0.342 ± 0.012	Tweedie	0.320 ± 0.011
Linear SVR	0.287 ± 0.005	kNeighbors	0.255 ± 0.001

REFERENCES

- [1] B. ADCOCK, C. BOYER, AND S. BRUGIAPAGLIA, *On oracle-type local recovery guarantees in compressed sensing*, Information and Inference: A Journal of the IMA, 10 (2020), pp. 1–49, <https://doi.org/10.1093/imaia/iaaa007>.
- [2] B. ADCOCK AND A. C. HANSEN, *Generalized sampling and infinite-dimensional compressed sensing*, Foundations of Computational Mathematics, 16 (2016), pp. 1263–1323, <https://doi.org/10.1007/s10208-015-9276-6>.
- [3] B. ADCOCK AND D. HUYBRECHS, *Approximating smooth, multivariate functions on irregular domains*, Forum of Mathematics, Sigma, 8 (2020), p. e26, <https://doi.org/10.1017/fms.2020.23>.
- [4] G. A. BARNETT, *A Robust RBF-FD Formulation based on Polyharmonic Splines and Polynomials*, PhD thesis, University of Colorado Boulder, 2015, <https://www.proquest.com/openview/16dabe0f8eb27e9679d09bb9c8eafbaa/1?pq-origsite=gscholar&cbl=18750>.
- [5] R. BASRI, M. GALUN, A. GEIFMAN, D. JACOBS, Y. KASTEN, AND S. KRITCHMAN, *Frequency bias in neural networks for input of non-uniform density*, in Proceedings of the 37th International Conference on Machine Learning, vol. 119, PMLR, 13–18 Jul 2020, pp. 685–694, <https://proceedings.mlr.press/v119/basri20a.html>.
- [6] V. BAYONA, N. FLYER, AND B. FORNBERG, *On the role of polynomials in rbf-fd approximations: Iii. behavior near domain boundaries*, Journal of Computational Physics, 380 (2019), pp. 378–399, <https://doi.org/10.1016/j.jcp.2018.09.029>.
- [7] C. BECK, S. BECKER, P. GROHS, N. JAAFARI, AND A. JENTZEN, *Solving the kolmogorov pde by means of deep learning*, Journal of Scientific Computing, 88 (2021), pp. 1–28, <https://doi.org/10.1007/s10915-021-01590-0>.
- [8] I. BOGAERT, B. MICHIELS, AND J. FOSTIER, *$O(1)$ computation of legendre polynomials and gauss–legendre nodes and weights for parallel computing*, SIAM Journal on Scientific Computing, 34 (2012), pp. C83–C101, <https://doi.org/10.1137/110855442>.
- [9] A. BUHR AND M. OHLBERGER, *Interactive simulations using localized reduced basis methods*, IFAC-PapersOnLine, 48 (2015), pp. 729–730, <https://doi.org/10.1016/J.IFACOL.2015.05.134>.
- [10] H. BUNGARTZ AND S. DIRNSTORFER, *Multivariate quadrature on adaptive sparse grids*, Computing, 71 (2003), pp. 89–114, <https://doi.org/10.1007/s00607-003-0016-4>.
- [11] Y. CHEN, J. JAKEMAN, C. J. GITTELSON, AND D. XIU, *Local polynomial chaos expansion for linear differential equations with high dimensional random inputs*, SIAM J. Sci. Comput., 37 (2015), <https://doi.org/10.1137/140970100>.
- [12] P. CHERIDITO, A. JENTZEN, AND F. ROSSMANNEK, *Efficient approximation of high-dimensional functions with deep neural networks*, ArXiv, abs/1912.04310 (2019), <https://arxiv.org/abs/1912.04310>.
- [13] G. G. CHRYSOS, M. GEORGOPOULOS, J. DENG, J. KOSSAIFI, Y. PANAGAKIS, AND A. ANANDKUMAR, *Augmenting deep classifiers with polynomial neural networks*, in European Conference on Computer Vision, Springer, 2022, pp. 692–716, https://doi.org/10.1007/978-3-031-19806-9_40.
- [14] G. G. CHRYSOS, S. MOSCHOGLOU, G. BOURITSAS, J. DENG, Y. PANAGAKIS, AND S. ZAFEIRIOU, *Deep polynomial neural networks*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 44 (2022), pp. 4021–4034, <https://doi.org/10.1109/TPAMI.2021.3058891>.
- [15] G. G. CHRYSOS, S. MOSCHOGLOU, G. BOURITSAS, Y. PANAGAKIS, J. DENG, AND S. ZAFEIRIOU, *P-nets: Deep polynomial neural networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 2180–2188, <https://doi.org/10.1109/CVPR42600.2020.00224>.

- [16] E. C. CYR, M. A. GULIAN, R. G. PATEL, M. PEREGO, AND N. A. TRASK, *Robust training and initialization of deep neural networks: An adaptive basis viewpoint*, in *Mathematical and Scientific Machine Learning*, PMLR, 2020, pp. 512–536, <https://proceedings.mlr.press/v107/cyr20a.html>.
- [17] P. J. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration*, Courier Corporation, 2007, <https://doi.org/10.1201/9780203713854>.
- [18] L. DEMANET AND L. YING, *Wave atoms and sparsity of oscillatory patterns*, *Applied and Computational Harmonic Analysis*, 23 (2007), pp. 368–387, <https://doi.org/10.1016/j.acha.2007.03.003>, <https://www.sciencedirect.com/science/article/pii/S1063520307000450>.
- [19] M. DOLBEAULT AND A. COHEN, *Optimal sampling and christoffel functions on general domains*, *Constructive Approximation*, 56 (2022), pp. 121–163, <https://doi.org/10.1007/s00365-021-09558-x>.
- [20] D. DUNG AND M. GRIEBEL, *Hyperbolic cross approximation in infinite dimensions*, *J. Complex.*, 33 (2015), pp. 55–88, <https://doi.org/10.1016/j.jco.2015.09.006>.
- [21] D. DUNG, V. TEMLYAKOV, AND T. ULLRICH, *Hyperbolic cross approximation*, *ArXiv: Numerical Analysis*, (2016), <https://doi.org/10.1007/978-3-319-92240-9>, <https://arxiv.org/abs/1604.08246>.
- [22] D. ELBRÄCHTER, D. PEREKRESTENKO, P. GROHS, AND H. BÖLCSKEI, *Deep neural network approximation theory*, *IEEE Transactions on Information Theory*, 67 (2021), pp. 2581–2623, <https://doi.org/10.1109/TIT.2021.3059134>.
- [23] G. E. FASSHAUER, *Meshfree approximation methods with MATLAB*, vol. 6, World Scientific, 2007, <https://doi.org/10.1142/6360>.
- [24] N. FLYER, B. FORNBERG, V. BAYONA, AND G. A. BARNETT, *On the role of polynomials in rbf-fd approximations: I. interpolation and accuracy*, *Journal of Computational Physics*, 321 (2016), pp. 21–38, <https://doi.org/10.1016/j.jcp.2016.05.026>.
- [25] W. GAUTSCHI, *Orthogonal polynomials: computation and approximation*, Oxford University Press, 2004, <https://doi.org/10.1093/acprof:oso/9780198506723.001.0001>.
- [26] T. GERSTNER AND M. GRIEBEL, *Numerical integration using sparse grids*, *Numerical Algorithms*, 18 (2004), pp. 209–232, <https://doi.org/10.1023/A:1019129717644>.
- [27] M. GOYAL, R. GOYAL, AND B. LALL, *Improved polynomial neural networks with normalised activations*, in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8, <https://doi.org/10.1109/IJCNN48605.2020.9207535>.
- [28] E. HABER AND L. RUTHOTTO, *Stable architectures for deep neural networks*, *Inverse Problems*, 34 (2017), <https://doi.org/10.1088/1361-6420/aa9a90>.
- [29] Z. HU, K. SHUKLA, G. KARNIADAKIS, AND K. KAWAGUCHI, *Tackling the curse of dimensionality with physics-informed neural networks*, *ArXiv, abs/2307.12306* (2023), <https://doi.org/10.48550/arXiv.2307.12306>.
- [30] S. IOFFE AND C. SZEGEDY, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, eds., vol. 37 of *Proceedings of Machine Learning Research*, Lille, France, 07–09 Jul 2015, PMLR, pp. 448–456, <https://doi.org/10.48550/arXiv.1502.03167>.
- [31] J. D. JAKEMAN, A. NARAYAN, AND T. ZHOU, *A generalized sampling and preconditioning scheme for sparse approximation of polynomial chaos expansions*, *SIAM Journal on Scientific Computing*, 39 (2017), pp. A1114–A1144, <https://doi.org/10.1137/16M1061197>.
- [32] A. JENTZEN, D. SALIMOVA, AND T. WELTI, *A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients*, *ArXiv, abs/1809.07321* (2018), <https://doi.org/10.4310/CMS.2021.v19.n5.a1>.
- [33] J. JIANG, Y. CHEN, AND A. NARAYAN, *A unified, goal-oriented, hybridized reduced basis method and generalized polynomial chaos algorithm for partial differential equations with random inputs*, *ArXiv: Numerical Analysis*, (2016), <https://arxiv.org/abs/1611.01385>.
- [34] J. JUMPER, R. EVANS, A. PRITZEL, T. GREEN, M. FIGURNOV, O. RONNEBERGER, K. TUNYASUVUNAKOOL, R. BATES, A. ŽÍDEK, A. POTAPENKO, ET AL., *Highly accurate protein structure prediction with alphafold*, *Nature*, 596 (2021), pp. 583–589, <https://doi.org/10.1038/s41586-021-03819-2>.
- [35] J. KILEEL, M. TRAGER, AND J. BRUNA, *On the expressive power of deep polynomial neural networks*, *Advances in Neural Information Processing Systems*, 32 (2019), <https://proceedings.neurips.cc/paper/2019/file/43c187ff0fa44a9be02c3f25ec7e5d5c-Paper.pdf>.
- [36] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in *3rd International Conference on Learning Representations, ICLR Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds., 2015, <https://doi.org/10.48550/arXiv.1412.6980>, <http://arxiv.org/>

- abs/1412.6980.
- [37] A. KRISHNAPRIYAN, A. GHOLAMI, S. ZHE, R. KIRBY, AND M. W. MAHONEY, *Characterizing possible failure modes in physics-informed neural networks*, Advances in Neural Information Processing Systems, 34 (2021), pp. 26548–26560, <https://doi.org/10.48550/arXiv.2011.05526>.
 - [38] R. LAM, A. SANCHEZ-GONZALEZ, M. WILLSON, P. WIRNSBERGER, M. FORTUNATO, F. ALET, S. RAVURI, T. EWALDS, Z. EATON-ROSEN, W. HU, ET AL., *Learning skillful medium-range global weather forecasting*, Science, 382 (2023), pp. 1416–1421, <https://doi.org/10.1126/science.adi2336>.
 - [39] H. LAROCHELLE, Y. BENGIO, J. LOURADOUR, AND P. LAMBLIN, *Exploring strategies for training deep neural networks*, J. Mach. Learn. Res., 10 (2009), pp. 1–40, <https://doi.org/10.5555/1577069.1577070>.
 - [40] D. LAUVERGNAT AND A. NAUTS, *Quantum dynamics with sparse grids: a combination of smolyak scheme and cubature. application to methanol in full dimensionality.*, Spectrochimica acta. Part A, Molecular and biomolecular spectroscopy, 119 (2014), pp. 18–25, <https://doi.org/10.1016/j.saa.2013.05.068>.
 - [41] M. LESHNO, V. Y. LIN, A. PINKUS, AND S. SCHOCKEN, *Multilayer feedforward networks with a non-polynomial activation function can approximate any function*, Neural Networks, 6 (1993), pp. 861–867, [https://doi.org/10.1016/S0893-6080\(05\)80131-5](https://doi.org/10.1016/S0893-6080(05)80131-5).
 - [42] B. LI, S. TANG, AND H. YU, *Better approximations of high dimensional smooth functions by deep neural networks with rectified power units*, arXiv preprint arXiv:1903.05858, (2019), <https://doi.org/10.48550/arXiv.1903.05858>.
 - [43] B. LI, S. TANG, AND H. YU, *Powernet: Efficient representations of polynomials and smooth functions by deep neural networks with rectified power units*, arXiv preprint arXiv:1909.05136, (2019), <https://arxiv.org/abs/1909.05136>.
 - [44] C. LI AND B. ADCOCK, *Compressed sensing with local structure: uniform recovery guarantees for the sparsity in levels class*, ArXiv, abs/1601.01988 (2016), <https://doi.org/10.1016/J.ACHA.2017.05.006>, <https://arxiv.org/abs/1601.01988>.
 - [45] S. LI AND J. BOYD, *Approximation on non-tensor domains including squircles, part iii: Polynomial hyperinterpolation and radial basis function interpolation on chebyshev-like grids and truncated uniform grids*, J. Comput. Phys., 281 (2015), pp. 653–668, <https://doi.org/10.1016/j.jcp.2014.10.035>.
 - [46] C. LIU, L. ZHU, AND M. BELKIN, *Loss landscapes and optimization in over-parameterized non-linear systems and neural networks*, Applied and Computational Harmonic Analysis, 59 (2022), pp. 85–116, <https://doi.org/10.1016/j.acha.2022.03.014>.
 - [47] D. C. LIU AND J. NOCEDAL, *On the limited memory bfgs method for large scale optimization*, Mathematical Programming, 45 (1989), pp. 503–528, <https://doi.org/10.1007/BF01589116>.
 - [48] Z. LIU, G. CHEN, Z. LI, S. QU, A. KNOLL, AND C. JIANG, *D2iftn: Disentangled domain-invariant feature learning networks for domain generalization*, IEEE Transactions on Cognitive and Developmental Systems, 15 (2023), pp. 2269–2281, <https://doi.org/10.1109/TCDS.2023.3264615>.
 - [49] I. LOSHCHELOV AND F. HUTTER, *Sgdr: Stochastic gradient descent with warm restarts*, in International Conference on Learning Representations, 2016, <https://doi.org/10.48550/arXiv.1608.03983>, <https://arxiv.org/abs/1608.03983>.
 - [50] Q. MENG, J. MATTHEW, V. ZIMMER, A. GÓMEZ, D. LLOYD, D. RUECKERT, AND B. KAINZ, *Mutual information-based disentangled neural networks for classifying unseen categories in different domains: Application to fetal ultrasound imaging*, IEEE Transactions on Medical Imaging, 40 (2020), pp. 722–734, <https://doi.org/10.1109/TMI.2020.3035424>.
 - [51] G. MIGLIORATI AND F. NOBILE, *Analysis of discrete least squares on multivariate polynomial spaces with evaluations at low-discrepancy point sets*, J. Complex., 31 (2015), pp. 517–542, <https://doi.org/10.1016/j.jco.2015.02.001>.
 - [52] V. NAIR AND G. E. HINTON, *Rectified linear units improve restricted boltzmann machines*, in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814, <http://www.csri.utoronto.ca/~hinton/absps/reluICML.pdf>.
 - [53] A. NARAYAN, J. JAKEMAN, AND T. ZHOU, *A christoffel function weighted least squares algorithm for collocation approximations*, Mathematics of Computation, 86 (2017), pp. 1913–1947, <https://doi.org/10.1090/mcom/3189>.
 - [54] J. A. OPSCHOOR, P. C. PETERSEN, AND C. SCHWAB, *Deep relu networks and high-order finite element methods*, Analysis and Applications, 18 (2020), pp. 715–770, <https://doi.org/10.1142/S0219530520500202>.
 - [55] J. A. OPSCHOOR AND C. SCHWAB, *Deep relu networks and high-order finite element methods*

- ii: *Chebyshev emulation*, arXiv preprint arXiv:2310.07261, (2023), <https://arxiv.org/abs/2310.07261>.
- [56] R. K. PACE AND R. BARRY, *Sparse spatial autoregressions*, Statistics and Probability Letters, 33 (1997), pp. 291–297, [https://doi.org/10.1016/S0167-7152\(96\)00140-X](https://doi.org/10.1016/S0167-7152(96)00140-X).
- [57] R. PASCANU, T. MIKOLOV, AND Y. BENGIO, *Understanding the exploding gradient problem*, ArXiv, abs/1211.5063 (2012), <https://arxiv.org/abs/1211.5063>.
- [58] A. PASZKE, S. GROSS, F. MASSA, A. LERER, J. BRADBURY, G. CHANAN, T. KILLEEN, Z. LIN, N. GIMELSHEIN, L. ANTIGA, ET AL., *Pytorch: An imperative style, high-performance deep learning library*, 2019, <https://pytorch.org>.
- [59] K. PETRAS, *Smolyak cubature of given polynomial degree with few nodes for increasing dimension*, Numerische Mathematik, 93 (2003), pp. 729–753, <https://doi.org/10.1007/s002110200401>.
- [60] N. RAHAMAN, A. BARATIN, D. ARPIT, F. DRAXLER, M. LIN, F. HAMPRECHT, Y. BENGIO, AND A. COURVILLE, *On the spectral bias of neural networks*, in International Conference on Machine Learning, PMLR, 2019, pp. 5301–5310, <https://proceedings.mlr.press/v97/rahaman19a.html>.
- [61] M. RAISSI, *Deep hidden physics models: Deep learning of nonlinear partial differential equations*, The Journal of Machine Learning Research, 19 (2018), pp. 932–955, <http://www.jmlr.org/papers/v19/18-046.html>.
- [62] 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 (2019), pp. 686–707, <https://doi.org/10.1016/j.jcp.2018.10.045>.
- [63] V. SHANKAR, *The overlapped radial basis function-finite difference (rbf-fd) method: A generalization of rbf-fd*, Journal of Computational Physics, 342 (2017), pp. 211–228, <https://doi.org/10.1016/j.jcp.2017.04.037>, <https://www.sciencedirect.com/science/article/pii/S0021999117303169>.
- [64] V. SHANKAR, R. M. KIRBY, AND A. L. FOGELSON, *Robust node generation for mesh-free discretizations on irregular domains and surfaces*, SIAM Journal on Scientific Computing, 40 (2018), pp. A2584–A2608, <https://doi.org/10.1137/17M114090X>.
- [65] V. SHANKAR, G. B. WRIGHT, AND A. L. FOGELSON, *An efficient high-order meshless method for advection-diffusion equations on time-varying irregular domains*, Journal of Computational Physics, 445 (2021), p. 110633, <https://doi.org/10.1016/j.jcp.2021.110633>, <https://www.sciencedirect.com/science/article/pii/S0021999121005283>.
- [66] G. SHEN, Y. JIAO, Y. LIN, AND J. HUANG, *Differentiable neural networks with repu activation: with applications to score estimation and isotonic regression*, arXiv preprint arXiv:2305.00608, (2023), <https://doi.org/10.48550/arXiv.2305.00608>.
- [67] J. SHEN AND L. WANG, *Sparse spectral approximations of high-dimensional problems based on hyperbolic cross*, SIAM J. Numer. Anal., 48 (2010), pp. 1087–1109, <https://doi.org/10.1137/090765547>.
- [68] T. SHIMOYAMA AND K. YOKOYAMA, *Localization and primary decomposition of polynomial ideals*, J. Symb. Comput., 22 (1996), pp. 247–277, <https://doi.org/10.1006/jsco.1996.0052>.
- [69] N. SRIVASTAVA, G. E. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, *Dropout: a simple way to prevent neural networks from overfitting*, J. Mach. Learn. Res., 15 (2014), pp. 1929–1958, <https://doi.org/10.5555/2627435.2670313>.
- [70] S. WANG, H. WANG, AND P. PERDIKARIS, *On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks*, Computer Methods in Applied Mechanics and Engineering, 384 (2021), p. 113938, <https://doi.org/10.1016/j.cma.2021.113938>.
- [71] V. WINSCHERL AND M. KRÄTZIG, *Solving, estimating, and selecting nonlinear dynamic models without the curse of dimensionality*, Econometrica, 78 (2010), pp. 803–821, <https://doi.org/10.3982/ecta6297>.
- [72] Z.-Q. J. XU, Y. ZHANG, T. LUO, Y. XIAO, AND Z. MA, *Frequency principle: Fourier analysis sheds light on deep neural networks*, arXiv preprint arXiv:1901.06523, (2019), <https://arxiv.org/abs/1901.06523>.
- [73] S. YAN AND J. JIN, *A dynamic p-adaptive dgtd algorithm for electromagnetic and multiphysics simulations*, IEEE Transactions on Antennas and Propagation, 65 (2017), pp. 2446–2459, <https://doi.org/10.1109/TAP.2017.2676724>.
- [74] L. ZHANG, J. HAN, H. WANG, R. CAR, AND E. WEINAN, *Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics*, Physical Review Letters, 120 (2018), p. 143001, <https://doi.org/10.1103/PhysRevLett.120.143001>.