An Extreme Learning Machine-Based Method for Computational PDEs in Higher Dimensions

Yiran Wang, Suchuan Dong*

Center for Computational & Applied Mathematics
Department of Mathematics
Purdue University, USA

(October 28, 2023)

Abstract

We present two effective methods for solving high-dimensional partial differential equations (PDE) based on randomized neural networks. Motivated by the universal approximation property of this type of networks, both methods extend the extreme learning machine (ELM) approach from low to high dimensions. With the first method the unknown solution field in d dimensions is represented by a randomized feed-forward neural network, in which the hidden-layer parameters are randomly assigned and fixed while the output-layer parameters are trained. The PDE and the boundary/initial conditions, as well as the continuity conditions (for the local variant of the method), are enforced on a set of random interior/boundary collocation points. The resultant linear or nonlinear algebraic system, through its least squares solution, provides the trained values for the network parameters. With the second method the high-dimensional PDE problem is reformulated through a constrained expression based on an Approximate variant of the Theory of Functional Connections (A-TFC), which avoids the exponential growth in the number of terms of TFC as the dimension increases. The free field function in the A-TFC constrained expression is represented by a randomized neural network and is trained by a procedure analogous to the first method. We present ample numerical simulations for a number of high-dimensional linear/nonlinear stationary/dynamic PDEs to demonstrate their performance. These methods can produce accurate solutions to high-dimensional PDEs, in particular with their errors reaching levels not far from the machine accuracy for relatively lower dimensions. Compared with the physics-informed neural network (PINN) method, the current method is both cost-effective and more accurate for high-dimensional PDEs.

Key words: high-dimensional PDE, extreme learning machine, randomized neural network, deep neural network, scientific machine learning, deep learning

1 Introduction

This work concerns the numerical approximation of partial differential equations (PDEs) in higher dimensions (typically beyond three). Mathematical models describing natural and physical processes or phenomena are usually expressed in PDEs. In a number of fields and domains, including physics, biology and finance, the models are naturally formulated in terms of high-dimensional PDEs. Well-known examples include the Schrodinger equation for many-body problems in quantum mechanics, the Black-Scholes equation for the price evolution of financial derivatives, and the Hamilton-Jacobi-Bellman (HJB) equation in dynamic programming and game theory [32, 80]. Development of computational techniques for PDEs is a primary thrust in scientific computing. In low dimensions, traditional numerical methods such as the finite difference, finite element (FEM), finite volume, and spectral type methods (and their variants), which are typically grid- or mesh-based, have achieved a tremendous success and are routinely used in computational science

*Author of correspondence. Emails: wang2335@purdue.edu (Y. Wang), sdong@purdue.edu (S. Dong)

and engineering applications. For high-dimensional PDEs, on the other hand, these mesh-based approaches encounter severe challenges owing to the curse of dimensionality, because the computational effort/complexity involved therein grows exponentially with increasing problem dimension [5, 13, 40, 33].

In the past few years deep neural networks (DNN or NN) have emerged as a promising approach to alleviate or overcome the curse of dimensionality for solving high-dimensional PDEs [4, 6, 39, 45]. DNN-based methods usually compute the PDE solution in a mesh-free manner by transforming the PDE problem into an optimization problem. The PDE and the boundary/initial conditions are encoded into the loss function by penalizing their residual norms on a set of sampling points. The differential operators involved therein are typically computed by automatic differentiation. The loss function is minimized by an optimizer, usually based on some flavor of gradient descent type algorithms [34]. Early works on NN-based methods for differential equations can be traced to the 1990s (see e.g. [50, 64, 14]). More recent prominent methods in this area include the physics-informed neural network (PINN) method [78], deep Galerkin method (DGM) [84], deep Ritz method [25], deep Nitsche method [55], deep mixed residual method [63], as well as other related approaches, variants and extensions (see e.g. [98, 67, 52, 44, 43, 12, 88, 87, 46, 60, 48, 93, 17, 68, 61, 1, 74], among others). Another approach for solving high-dimensional PDEs is to reformulate the problem using stochastic differential equations, thus casting the PDE problem into a learning problem. Representative techniques of this type include the deep backward stochastic differential equation (Deep BSDE) [24, 32] and the forward-backward stochastic neural network method [77]. Temporal difference learning has been employed in [95, 59] for solving high-dimensional parabolic PDEs and partial integro-differential equations, which discretizes the problem in time and represents the solution by a neural network at each time step. A data-driven method is developed in [68] to approximate the semi-global solutions to the HJB equations for high-dimensional nonlinear systems and to compute the optimal feedback controls. In [61] the generalization error bounds are derived for two-layer neural networks in the framework of deep Ritz method for solving two elliptic PDEs, and it is shown that the errors are independent of the problem dimension. We would also like to refer the reader to [90] for a recent review of NN-based techniques for high-dimensional PDEs.

For the neural network-based techniques reviewed above for high-dimensional PDEs, all the weight/bias parameters in the neural network are trained and determined by an optimizer, which in most cases is Adam, L-BFGS or some related variant. Unlike these methods, in the current work we consider another type of neural networks for the computation of high-dimensional PDEs, referred to as randomized neural networks (or random-weight neural networks), in which a subset of the network parameters is assigned to random values and fixed (not trainable) while the rest of the network parameters are trained.

Randomness has long been exploited in neural networks [81]. Randomized neural networks can be traced to the un-organized machine by Turing [89] and the perceptron by Rosenblatt [79] in the 1950s. Since the early 1990s, methods based on randomized NNs have witnessed a strong resurgence and expansion [85, 11], with prominent techniques widely applied and exerting a profound influence over a variety of areas [81, 29].

A simple strategy underlies randomized neural networks. Since it is extremely hard and expensive to optimize the full set of weight/bias parameters in the neural network, it seems sensible if a subset of the network parameters is randomly assigned and fixed, so that the resultant optimization problem of network training can become simpler, and in certain cases linear, hopefully without severely sacrificing the network's achievable approximation capacity [21, 70]. When applied to different types of neural networks or under different configurations, randomization gives rise to several techniques, including the random vector functional link (RVFL) network [73, 72, 41], the extreme learning machine (ELM) [37, 38], and the echo-state network [42, 62], among others.

We consider the extreme learning machine (ELM) approach for high-dimensional PDE problems. The original work on ELM was [36, 37], developed for linear classification and regression problems with single hidden-layer feed-forward neural networks. This method has since found widespread applications in many fields [35, 2]. ELM is characterized by two ideas, randomly-assigned non-trainable (fixed) hidden-layer parameters, and trainable linear output-layer parameters determined by linear least squares method or by the pseudo-inverse of coefficient matrix [72, 8]. Randomized neural networks of the ELM type and its close cousin RVFL type, with a single hidden layer, are universal function approximators. Their universal approximation ability has been established by the theoretical studies of [41, 51, 38, 69]. In particular, the expected rate of convergence for approximating Lipschitz continuous functions has been provided by [41, 76, 69] (see also Section 2.2 below).

The adoption of ELM for scientific computing, in particular for the numerical solution of differential

equations, occurs only fairly recently. The existing works in this area have been confined to PDEs in low dimensions (primarily one or two spacial dimensions) or ordinary differential equations (ODEs) so far. Early works in this regard [91, 86, 57] have used polynomials (e.g. Chebyshev, Legendre, Bernstein) as activation functions for solving linear ODEs/PDEs. Subsequent contributions have explored other types of functions and made advances on a variety of fronts. While many studies are confined to linear ODE/PDE problems (see e.g. [71, 22, 58, 10, 23, 54, 75, 9]), ELM-based methods for nonlinear PDEs/ODEs have been developed in e.g. [15, 16, 83, 26, 21, 82, 70, 19, 27, 28] (among others). As has become clear from these studies, the ELM technique can produce highly accurate solutions to linear and nonlinear PDEs in low dimensions (and ODEs) with a competitive computational cost. For smooth solutions the ELM errors decrease exponentially as the number of degrees of freedom (number of training points, or number of trainable parameters) increases [15, 70], reminiscent of the traditional high-order methods such as the spectral or spectral element techniques [47, 94, 97, 18, 92, 56]. Their errors can reach the level of machine accuracy as the degrees of freedom become large [21]. In the presence of local complex features (e.g. sharp gradient) in the solution field, a combination of domain decomposition and ELM, referred to as local ELM (or locELM) in [15], will be critical to achieving a high accuracy [70]. ELM-based methods have been compared extensively with the traditional numerical methods (e.g. classical FEM, high-order finite elements) and with the dominant DNN-based solvers (e.g. PINN/DGM) for low-dimensional PDE problems; see e.g. [21, 15]. ELM far outperforms the classical FEM, and also outperforms the high-order FEM markedly when the problem size is not very small [21]. With a small problem size, the performance of ELM and high-order FEM is comparable, with the latter being slightly better [21]. Here "outperform" refers to the ability of a method to achieve a better accuracy under the same computational cost or to incur a lower computational cost for the same accuracy. ELM also considerably outperforms DGM and PINN for low-dimensional problems [15]. Very recently it has been shown by [19] that the ELM-based method exhibits a spectral accuracy for solving inverse PDE problems (in low dimensions) if the measurement data is noise-free, when the network is trained by nonlinear least squares or the variable projection algorithm [20].

In the current paper we focus on the computation of high-dimensional PDEs with the ELM-based approach. There seems to be very little investigation in this aspect so far. A recent work related to this topic is [30], in which random feature neural networks are found to be able to approximate the functions with a convolutional structure efficiently (without curse of dimensionality). Since the solutions to linear Kolmogorov PDEs associated to exponential Levy models (e.g. Black-Scholes equation) can be expressed into this type of functions based on the Feynman-Kac formula, this work recasts the problem of learning the solution to linear Kolmogorov PDEs into a regression problem and employs random feature neural networks to approximate the solution data. We note that the method in [30] relies on an external Monte-Carlo solver to first generate the solution data to the Kolmogorov PDE in order to train the random feature neural network by regression. We also note the interesting theoretical aspect of [30].

The technique developed in the current work computes high-dimensional PDEs in a "physics-informed" manner, which is self-contained and does not rely on any external PDE solver. The high-dimensional initial/boundary value problems considered here also involve more general PDEs and bounded domains. We are especially interested in the following question:

- Is the ELM-type randomized neural network approach effective for computational PDEs in high dimensions?

The objective of this paper is to present two ELM-based methods for solving high-dimensional PDEs, and to demonstrate with numerical simulations that these methods provide a positive answer to the above question, at least for the range of problem dimensions studied in this paper.

The first method (termed simply ELM herein) extends the ELM technique and its local variant locELM developed in [15] (for low-dimensional problems) to linear and nonlinear PDEs in high dimensions. The solution field to the high-dimensional PDE problem is represented by a randomized feed-forward neural network, with its hidden-layer coefficients randomly assigned and fixed and its output-layer coefficients trained. Enforcing the PDE, the boundary and initial conditions on a random set of collocation points from the domain interior and domain boundaries gives rise to a linear or nonlinear algebraic system of equations about the trainable NN parameters. We seek a least squares solution to this algebraic system, attained by either linear or nonlinear least squares method, which provides the trained values for the network parameters. In the local variant of this method, the high-dimensional domain is decomposed along a maximum of $\mathcal{M}$ ($\mathcal{M} = 2$ herein) directions, and the solution field on each sub-domain is represented by an ELM-type

randomized neural network. We enforce the PDE, the boundary/initial conditions and appropriate continuity conditions across sub-domains on a set of random collocation points from each sub-domain, from the domain boundaries and from the shared sub-domain boundaries. The resultant linear or nonlinear algebraic system yields, by its least squares solution, the trained values for the network parameters of the local NNs.

The second method (termed ELM/A-TFC herein) combines the ELM approach and an approximate variant of the theory of functional connections (TFC) for solving high-dimensional PDEs. TFC [65, 66] provides a systematic approach for enforcing the boundary/initial conditions through a constrained expression (see e.g. [83, 49]). However, the number of terms in TFC constrained expressions grows exponentially with respect to the problem dimension, rendering TFC infeasible for high-dimensional problems. By noting a hierarchical decomposition of the constrained expression, we introduce an approximate variant of TFC (referred to as A-TFC herein) that retains only the dominant terms therein. A-TFC avoids the exponential growth in the number of terms of TFC and is suitable for high-dimensional problems. On the other hand, since A-TFC is an approximation of TFC, its constrained expression does not satisfy the boundary conditions unconditionally for an arbitrary free function contained therein. However, the conditions for the free function of the A-TFC constrained expression in general involve functions of a simpler form, which is effectively a linearized form of those of the original boundary/initial conditions. A-TFC represents a trade-off. It carries a level of benefit of TFC for enforcing the boundary/initial conditions and is simultaneously suitable for high-dimensional PDEs. The ELM/A-TFC method uses the A-TFC constrained expression to reformulate the given high-dimensional PDE problem into a transformed problem about the free function contained in the expression. This free function is then represented by an ELM-type randomized neural network, and the reformulated PDE problem is enforced on a set of random collocation points. The least squares solution to the resultant algebraic system provides the trained values for the network parameters, thus leading to the solution for the free function. The solution to the original high-dimensional PDE problem is then computed based on the A-TFC constrained expression.

Ample numerical simulations are presented to test these methods for a number of high-dimensional PDEs that are linear or nonlinear, stationary or time-dependent. The current method has also been compared with the PINN method for a range of problem dimensions. The numerical results show that the current methods exhibit a clear sense of convergence with respect to the number of training parameters and the number of boundary collocation points for high-dimensional PDEs. The rate of convergence is close to exponential for an initial range of parameter values (before saturation). These methods can capture the solutions to high-dimensional PDEs quite accurately, in particular with their errors reaching levels not far from the machine accuracy for comparatively lower dimensions. The error levels produced by these two methods are generally comparable, with ELM/A-TFC appearing slightly better in lower dimensions. On the other hand, ELM generally involves a smaller computational effort and cost than ELM/A-TFC. Compared with PINN, the current ELM method can achieve a significantly better accuracy under a markedly lower computational cost (network training time) for solving high-dimensional PDEs.

The contributions of this paper lie in the ELM method and the ELM/A-TFC method presented herein for computing high-dimensional PDE problems. To the best of the authors' knowledge, this seems to be the first physics-informed technique based on ELM-type randomized neural networks for solving high-dimensional PDEs.

The methods presented in this paper are implemented in Python based on the Tensorflow and Keras libraries. The linear and nonlinear least squares methods are based on routines from the Scipy library. The numerical simulations are performed on a MAC computer (Apple M1 Chip, 8 cores, 8GB memory, 250GB hard disk, macOS Ventura) in the authors' institution.

The rest of this paper is organized as follows. In Section 2 we first briefly recall the theoretical result on ELM-type randomized NNs for function approximations in high dimensions, and then describe the ELM method and the ELM/A-TFC method for solving high-dimensional PDEs. In Section 3 we present extensive numerical simulations to test these two methods with several linear and nonlinear, stationary and dynamic PDEs for a range of problem dimensions. The current method is also compared with PINN. Section 4 concludes the presentation with a summary of the results and some further remarks.

2 Extreme Learning Machine for High-Dimensional PDEs

Suppose $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_d$ is a domain in $\mathbb{R}^d$ (d being a positive integer) with boundary $\partial\Omega$, where $\Omega_i = [a_i, b_i]$ for given constants a_i and b_i ($1 \leqslant i \leqslant d$). We consider the boundary value problem below,

$$\mathcal{L}u(x) + \mu\mathcal{N}(u(x)) = Q(x), \quad x \in \Omega, \tag{1a}$$

$$\mathcal{B}u(x) = H(x), \quad x \in \partial\Omega. \tag{1b}$$

Here $\mathcal{L}$ and $\mathcal{N}$ are linear and nonlinear differential operators, respectively. $u(x) \in \mathbb{R}$ is the unknown field to be computed. $\mathcal{B}$ is a linear differential or algebraic operator, and equation (1b) represents the boundary conditions. Q and H are given functions, and μ is a constant. If $\mu = 0$, the problem is linear. We assume that $\mathcal{L}$ may contain time derivatives (e.g. $\frac{\partial}{\partial t}$ or $\frac{\partial^2}{\partial t^2}$, with t being the time variable). In this case, the problem is time dependent, and we will treat t in the same fashion as x. More specifically, we will treat this as a $(d+1)$-dimensional problem with t as the last dimension, $x = (x_1, \ldots, x_d, x_{d+1} = t)$, where x_i ($1 \leqslant i \leqslant d+1$) denotes the components of x. Accordingly, in this case we will assume that (1b) contains appropriate initial conditions with respect to t. The point here is that the problem (1) may represent an initial/boundary value problem, and we will not distinguish this case in the following discussions unless necessary.

In what follows we present two methods for solving the system (1). The first is an extension to high dimensions of the ELM technique originally developed in [15] for low-dimensional problems. The second method is a combination of ELM with an approximate variant of the Theory of Functional Connections, termed A-TFC, which avoids the exponential growth in the computational effort of TFC in high dimensions.

2.1 Randomized Feed-Forward Neural Networks

We consider the approximation of the solution field $u(x)$ to system (1) by a randomized feed-forward neural network. A feed-forward neural network (FNN) having $(L+1)$ ($L \geqslant 2$) layers represents a parameterized function $\mathcal{G}(x;\theta)$ given by (for the input x and parameter θ) [31],

$$\mathcal{G}(x;\theta) = W_L \cdot \sigma\left(W_{L-1} \cdot \sigma\left(\cdots \sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x + b_1\right) + b_2\right)\cdots\right) + b_{L-1}\right) + b_L, \tag{2}$$

where W_i and b_i ($1 \leqslant i \leqslant L$) are the weight and bias in the i-th layer, $\theta = (W_1, \ldots, W_L, b_1, \ldots, b_L)$, and $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function. Layer 0 (input layer) contains d nodes, representing the components of x, and layer L (output layer) contains a single node, representing $u(x)$. The layers in between are the hidden layers. Note that the output layer in (2) is linear, with no activation function applied. In the current paper we further assume that the output layer has zero bias, i.e. $b_L = 0$.

A randomized feed-forward neural network is an FNN in which a subset of the network parameters θ are assigned to random values and fixed (non-trainable), while only the rest of the network parameters are trained. Extreme learning machine (ELM) [37] is one type of randomized neural networks, in which all the hidden-layer coefficients are randomly assigned and fixed and only the output-layer coefficients are trained [15]. In the current work we approximate the solution field $u(x)$ to the system (1) by ELM, and assign the network coefficients in all the hidden layers to uniform random values from the interval $[-R_m, R_m]$, where R_m is a constant. We note that it is possible to assign the random parameters in other fashions, e.g. by fixing the hidden-layer weights/biases to ones or zeros and randomly assigning a shape parameter introduced in the activation function such as in [27].

Remark 1. *While multiple hidden layers can be employed in ELM networks, computational experience seems to indicate that a deeper network does not in general deliver better simulation results (they usually seem somewhat worse) compared with a shallow network with ELM. In practice, a small number of hidden layers (e.g. one, or perhaps two) are usually employed in the actual simulations with ELM; see e.g. [15] (among others). In the numerical simulations with high-dimensional PDEs in this paper, we have employed ELM networks with a single hidden layer (see Section 3).*

2.2 Randomized NNs for High-Dimensional Function Approximation

Extreme learning machines (or RVFL networks) are universal function approximators; see e.g. [41, 38]. The universal approximation theorems [41, 38] basically state that any given continuous function can be

approximated by a randomized NN having a single hidden layer, in which the hidden-layer coefficients are randomly assigned and fixed and the output-layer coefficients can be adjusted/trained, to any desired degree of accuracy, if the number of hidden units is sufficiently large.

We next recall the result from [41] concerning the convergence rate of randomized NNs for function approximations in high dimensions, which motivates the development of the current methods for high-dimensional computational PDEs.

Define $I^d := [0,1]^d$ and consider a continuous function $f \in C(I^d)$ satisfying the Lipschitz condition, i.e., there exists a constant $\gamma > 0$ such that for any $x, y \in I^d$,

$$|f(x) - f(y)| \leq \gamma \|x - y\|, \tag{3}$$

where $\|x - y\| := \sum_{i=1}^d |x_i - y_i|$. To approximate f, we construct a sequence functions $\{f_{\omega_n}\}$ as follows,

$$f_{\omega_n}(x) = \sum_{j=1}^n a_j g(w_j \cdot x + b_j), \tag{4}$$

where ω_n is defined as $\omega_n := (n, a_1, \ldots, a_n, b_1, \ldots, b_n, w_1, \ldots, w_n)$. In particular, $\lambda_n := \{b_1, \ldots, b_n, w_1, \ldots, w_n\}$ denotes a set of random parameters from some probabilistic space $S_n(\Omega, \alpha)$, where α is a parameter. The corresponding probability measure $\mu_{n,\Omega,\alpha}$ is specified as follows. Suppose $\hat{w}_0 = (\hat{w}_{01}, \ldots, \hat{w}_{0d})$, $y_0 = (y_{01}, \ldots, y_{0d})$ and u_0 are independent and uniformly distributed in $V^d = [0, \Omega] \times \cdots \times [-\Omega, \Omega]$, I^d and $[-2\Omega, 2\Omega]$, respectively. Then $w_0 = \alpha \hat{w}_0$ and $b_0 = -w_0 \cdot y_0 - u_0$. $(w_1, \ldots, w_n)$ and $(b_1, \ldots, b_n)$ are two sets of samples of the random variables w_0 and b_0. g in (4) is the activation function, chosen to be absolutely integrable, i.e.,

$$\int_{\mathbb{R}} g^2(x) dx < +\infty. \tag{5}$$

We further restrict g on some compact support $\prod_{i=1}^d [-\beta w_i, \beta w_i]$ to get g_β (β denotes a parameter). Then the following result holds.

Theorem 1. *[41] For any $f \in C(I^d)$ satisfying (3), any compact $K \subset I^d$ that is a proper subset of I^d, and any activation function g_β satisfying (5), there exists a sequence of $\{f_{\omega_n}\}$ and probability measure $\mu_{n,\Omega,\alpha}$ such that*

$$E \int_K |f(x) - f_{\omega_n}(x)|^2 dx \leq \frac{C_{f,g,\Omega,\alpha,\beta,d}}{n}, \tag{6}$$

for some constant $C_{f,g,\Omega,\alpha,\beta,d}$ independent of n.

Remark 2. *The theorem can be generalized when I^d is replaced by $[a,b]^d$ by a change of variables. We omit this detail and consider the generic situation.*

Remark 3. *It is notable that the approximation error is of the order $1/\sqrt{n}$ as the number of basis functions n increases, irrespective of the dimension d. This indicates that the approximation (4) with random basis functions can be effective for high dimensions. On the other hand, one notes from [3] that the approximation by linear combinations of deterministic and fixed bases leads to an approximation error on the order of $\mathcal{O}(1/n^{\frac{1}{d}})$. In other words, if deterministic and fixed basis functions are used, it is impossible to avoid the exponential growth in d for the number of basis functions. The randomized bases (such as in ELM and RVFL), however, can be effective for high-dimensional function approximations in the sense of the expectation.*

2.3 Solving High-Dimensional PDEs with ELM

Adopting ELM for computational PDEs is characterized by two ideas: (i) The hidden-layer coefficients are assigned to random values and fixed, and only the output-layer coefficients are trainable, as already mentioned previously. (ii) The trainable network parameters are determined by the linear or nonlinear least squares method [15], not by the gradient descent type algorithms. This means that in equation (2) the

coefficients (W_i, b_i) for $1 \leqslant i \leqslant L-1$ will be assigned to uniform random values from $[-R_m, R_m]$ and fixed, while only W_L is trained (noting that we set $b_L = 0$).

Let N_u denote the number of nodes in the last hidden layer of the neural network, and $V_j(x)$ $(1 \leqslant j \leqslant N_u)$ denote the output fields of the last hidden layer. Then equation (2) can be written into,

$$u(x) = \sum_{i=1}^{N_u} \phi_j V_j(x) = \mathbf{V}(x)\mathbf{\Phi}, \quad x \in \Omega, \tag{7}$$

where $\mathbf{V}(x) = (V_1(x), \ldots, V_{N_u}(x)) = \sigma\left(W_{L-1} \cdot \sigma\left(\cdots \sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x + b_1\right) + b_2\right) \cdots\right) + b_{L-1}\right)$, and $\mathbf{\Phi} = (\phi_1, \ldots, \phi_{N_u})^T = W_L$. Note that $\mathbf{V}(x)$ is fixed once the hidden-layer coefficients are randomly assigned and ϕ_j $(1 \leqslant j \leqslant N_u)$ are the output-layer coefficients (trainable parameters) of ELM.

The residual function of the system (1) is,

$$R(x, y, \mathbf{\Phi}) = \begin{bmatrix} R^{\mathrm{pde}}(x, \mathbf{\Phi}) \\ R^{\mathrm{bc}}(y, \mathbf{\Phi}) \end{bmatrix} = \begin{bmatrix} \mathcal{L}\mathbf{V}(x)\mathbf{\Phi} + \mu\mathcal{N}(\mathbf{V}(x)\mathbf{\Phi}) - Q(x), \ x \in \Omega \\ \mathcal{B}\mathbf{V}(y)\mathbf{\Phi} - H(y), \ y \in \partial\Omega \end{bmatrix} \tag{8}$$

where R^{pde} and R^{bc} are the residuals corresponding to the PDE and the boundary/initial conditions, respectively.

We next choose a set of collocation points from the domain interior and the domain boundaries, and enforce the residual function (8) to be zero on these collocation points. For solving high-dimensional PDEs we will employ a set of random collocation points from the interior/boundary of the domain in this work. We note that the regular grid points, which have been used with ELM in e.g. [15, 16, 21, 70, 19] as the sampling points for low-dimensional problems, are not feasible for high-dimensional PDEs, because of the exponential growth in the number of points with increasing dimension.

More specifically, the collocation points are set as follows. Let N_{in} denote the number of random collocation points in the interior of Ω, and N_{bc} denote the number of random collocation points on each hyperface of $\partial\Omega$. To generate the interior collocation points we set

$$P_i = (x_{1,i}, \ldots, x_{d,i}), \quad \text{for } i \in [N_{\mathrm{in}}], \tag{9}$$

where $x_{j,i} \in (a_j, b_j)$ is a uniform random value for $j \in [d]$. Here we have used the notation in combinatorics that $[d] = \{1, \ldots, d\}$. For the boundary collocation points, we choose N_{bc} random points on each hyperface of $\partial\Omega$, which is a $(d-1)$-dimensional hyperplane, and there are $2d$ hyperfaces in total. The total number of boundary collocation points is $N_{\mathrm{bc}}^{\mathrm{tot}} = 2dN_{\mathrm{bc}}$. For $i \in [N_{\mathrm{bc}}]$, $j \in [d]$ and $l \in \{0, 1\}$, we set

$$P_{i,j,l} = (x_{1,i}, \ldots, x_{j-1,i}, x_{j,i} = a_j\delta_{l,0} + b_j\delta_{l,1}, x_{j+1,i}, \ldots, x_{d,i}) \tag{10}$$

as the boundary collocation points, in which $x_{k,i} \in (a_k, b_k)$ is a uniform random value if $k \neq j$. Here $\delta_{i,j}$ denotes the Kronecker delta, $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise. Overall, the total number of collocation points is $N_c = N_{\mathrm{in}} + N_{\mathrm{bc}}^{\mathrm{tot}}$. Let $N_a = N_c + N_{\mathrm{bc}}^{\mathrm{tot}}$, and

$$\mathbf{x}_{\mathrm{in}} = \begin{bmatrix} \vdots \\ P_i \\ \vdots \end{bmatrix} \in \mathbb{M}^{N_{\mathrm{in}} \times d}, \quad \mathbf{y} = \begin{bmatrix} \vdots \\ P_{i,j,l} \\ \vdots \end{bmatrix} \in \mathbb{M}^{N_{\mathrm{bc}}^{\mathrm{tot}} \times d}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_{\mathrm{in}} \\ \mathbf{y} \end{bmatrix} \in \mathbb{M}^{N_c \times d}, \tag{11}$$

where $\mathbb{M}^{a \times b}$ denotes the set of matrices with shape $a \times b$.

Enforcing the residual function (8) to be zero on all the collocation points gives rise to the following system,

$$\mathbf{0} = \mathbf{R}(\mathbf{\Phi}) = R(\mathbf{x}, \mathbf{y}, \mathbf{\Phi}) = \begin{bmatrix} R^{\mathrm{pde}}(\mathbf{x}, \mathbf{\Phi}) \\ R^{\mathrm{bc}}(\mathbf{y}, \mathbf{\Phi}) \end{bmatrix} = \begin{bmatrix} \mathcal{L}\mathbf{V}(\mathbf{x})\mathbf{\Phi} + \mu\mathcal{N}(\mathbf{V}(\mathbf{x})\mathbf{\Phi}) - Q(\mathbf{x}) \\ \mathcal{B}\mathbf{V}(\mathbf{y})\mathbf{\Phi} - H(\mathbf{y}) \end{bmatrix} = \begin{bmatrix} \mathbf{R}^{\mathrm{pde}}(\mathbf{\Phi}) \\ \mathbf{R}^{\mathrm{bc}}(\mathbf{\Phi}) \end{bmatrix}. \tag{12}$$

Here $\mathbf{R}(\mathbf{\Phi}) \in \mathbb{M}^{N_a \times 1}$, $\mathbf{R}^{\mathrm{pde}}(\mathbf{\Phi}) \in \mathbb{M}^{N_c \times 1}$, $\mathbf{R}^{\mathrm{bc}}(\mathbf{\Phi}) \in \mathbb{M}^{N_{\mathrm{bc}}^{\mathrm{tot}} \times 1}$, $\mathbf{V}(\mathbf{x}) \in \mathbb{M}^{N_c \times N_u}$, and $\mathbf{V}(\mathbf{y}) \in \mathbb{M}^{N_{\mathrm{bc}}^{\mathrm{tot}} \times N_u}$.

The system (12) is an algebraic system about $\mathbf{\Phi}$, containing N_a equations with N_u unknowns. We seek a least squares solution to this system. When $\mu = 0$, this system is linear about $\mathbf{\Phi}$ and can be written as

$$\begin{bmatrix} \mathcal{L}\mathbf{V}(\mathbf{x}) \\ \mathcal{B}\mathbf{V}(\mathbf{y}) \end{bmatrix} \mathbf{\Phi} = \begin{bmatrix} Q(\mathbf{x}) \\ H(\mathbf{y}) \end{bmatrix}. \tag{13}$$

In this case we compute $\mathbf{\Phi}$ by solving the system (13) using the linear least squares method (with a minimum norm if the coefficient matrix is rank deficient) [7].

When $\mu \neq 0$, the algebraic system (12) is nonlinear with respect to $\mathbf{\Phi}$. In this case we compute $\mathbf{\Phi}$ by solving this system using the nonlinear least squares method with perturbations (NLLSQ-perturb) from [15, 21, 19]. The nonlinear least squares method [7] as in [15, 21, 19] represents a Gauss-Newton method combined with a trust-region strategy. The NLLSQ-perturb algorithm requires, for an arbitrary given $\mathbf{\Phi} \in \mathbb{R}^{N_u}$, the computation of $\mathbf{R}(\mathbf{\Phi})$ and its Jacobian matrix $\mathbf{J}(\mathbf{\Phi}) = \frac{\partial \mathbf{R}}{\partial \mathbf{\Phi}} \in \mathbb{M}^{N_a \times N_u}$ for the Gauss-Newton iterations (see [15, 21] for details). The Jacobian matrix for (12) is given by

$$\mathbf{J}(\mathbf{\Phi}) = \begin{bmatrix} \mathcal{L}\mathbf{V}(\mathbf{x}) + \mu \mathcal{N}'(\mathbf{V}(\mathbf{x})\mathbf{\Phi})\mathbf{V}(\mathbf{x}) \\ \mathcal{B}\mathbf{V}(\mathbf{y}) \end{bmatrix}, \tag{14}$$

where $\mathcal{N}'(u) = \frac{\partial \mathcal{N}}{\partial u}$.

Remark 4. *In our implementation, the input data to the ELM neural network consists of* $\mathbf{x}$ *(coordinates of all collocation points), and the output data consists of* $u(\mathbf{x}) \in \mathbb{M}^{N_c \times 1}$ *(the solution field evaluated on the collocation points). After the hidden-layer coefficients are randomly assigned,* $V(\mathbf{x})$ *is computed by a forward evaluation of a sub-network, implemented in Keras as a sub-model of the original NN, whose input is* $\mathbf{x}$ *and output is the last hidden layer of the original NN. The differential operators involved in* $\mathcal{L}V(\mathbf{x})$ *and* $\mathcal{B}V(\mathbf{x})$ *are then computed by a forward-mode automatic differentiation with this sub-model. The linear least squares method is based on the routine scipy.linalg.lstsq from the SciPy library. The NLLSQ-perturb algorithm for the nonlinear least squares method is based on the routine scipy.optimize.least_ squares from the SciPy library (see [15] or the Appendix A of [19] for more details).*

Remark 5. *When the solution field contains local features (e.g. sharp gradient), it would be preferable to combine ELM with domain decomposition for approximating the solution, thus leading to the local ELM approach (called locELM in [15]). In this case, we represent the solution on each sub-domain by a local randomized FNN, and impose* C^k *(with* k *related to the PDE order) continuity conditions across the shared sub-domain boundaries.*

For high-dimensional PDE problems, if domain decomposition is performed in every direction, the number of sub-domains would increase exponentially with increasing problem dimension. To avoid the exponential growth in the number of sub-domains in local ELM, we require that the domain should be decomposed only along a maximum of $\mathcal{M}$ *directions, where* $\mathcal{M}$ *is a fixed small integer (*$0 \leqslant \mathcal{M} \leqslant d$*). The specific directions in which the domain is decomposed can be any subset of the* d *directions with a size not exceeding* $\mathcal{M}$*. We use* $\mathcal{M} = 2$ *in the current work, i.e. domain decomposition in a maximum of two directions with local ELM.*

In the presence of domain decomposition, the residual function (8) *for the system* (1) *needs to be modified accordingly to account for the* C^k *continuity conditions across sub-domain boundaries. Let* N_Ω *denote the number of sub-domains, and* Ω_i *(*$1 \leqslant i \leqslant N_\Omega$*) denote the* i*-th sub-domain. Symbolically, the modified residual function can be written as,*

$$R(x, y, z, \mathbf{\Phi}) = \begin{bmatrix} R^{pde}(x, \mathbf{\Phi}) \\ R^{bc}(y, \mathbf{\Phi}) \\ R^{ck}(z, \mathbf{\Phi}) \end{bmatrix} = \begin{bmatrix} \mathcal{L}\mathbf{V}_i(x)\mathbf{\Phi}_i + \mu \mathcal{N}(\mathbf{V}_i(x)\mathbf{\Phi}_i) - Q(x),\ x \in \Omega_i,\ 1 \leqslant i \leqslant N_\Omega \\ \mathcal{B}\mathbf{V}_i(y)\mathbf{\Phi}_i - H(y),\ y \in \partial\Omega_i \cap \partial\Omega,\ 1 \leqslant i \leqslant N_\Omega \\ \mathcal{C}\mathbf{V}_i(z)\mathbf{\Phi}_i - \mathcal{C}\mathbf{V}_j(z)\mathbf{\Phi}_j,\ z \in \partial\Omega_i \cap \partial\Omega_j,\ \forall\ \text{adjacent } (\Omega_i, \Omega_j) \end{bmatrix}, \tag{15}$$

where $\mathbf{V}_i(x) = (V_{i1}(x), \ldots, V_{iN_u}(x))$ *denotes the output fields of the last hidden layer of the local NN on* Ω_i*,* $\mathbf{\Phi}_i = (\phi_{i1}, \ldots \phi_{iN_u})^T$ *denotes the vector of output-layer coefficients of the local network for* Ω_i*, and* $\mathbf{\Phi} = (\mathbf{\Phi}_1^T, \ldots, \mathbf{\Phi}_{N_\Omega}^T)^T$ *denotes the set of training parameters of the overall problem. The solution field* $u(x)$*, when restricted to* Ω_i *(*$1 \leqslant i \leqslant N_\Omega$*), is given by*

$$u_i(x) = \sum_{j=1}^{N_u} \phi_{ij} V_{ij}(x) = \mathbf{V}_i(x)\mathbf{\Phi}_i, \quad x \in \Omega_i. \tag{16}$$

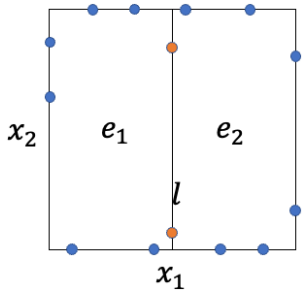


Figure 1: Illustration of random boundary collocation points on two adjacent sub-domains in two dimensions.

In (15), $R^{ck}(z, \mathbf{\Phi}) = \mathcal{C}u_i(z) - \mathcal{C}u_j(z)$ *denotes the residual corresponding to the continuity conditions between* $u_i(z)$ *and* $u_j(z)$ *on their shared sub-domain boundary, and* $\mathcal{C}$ *denotes a differential (or algebraic) operator corresponding to the continuity conditions. If the PDE order is* m $(m \geqslant 1)$ *with respect to* x_i*, we will in general impose* C^{m-1} *conditions across the sub-domain boundaries along the* x_i *direction.*

Accordingly, we choose a set of random collocation points on the interior and on the boundaries of each sub-domain, and enforce the residual function (15) *to be zero on these collocation points. This leads to a linear or nonlinear algebraic system about* $\mathbf{\Phi}$*, which is solved by the linear or nonlinear least squares method to attain a least squares solution for the training parameters* $\mathbf{\Phi}$*.*

Let us next further comment on enforcing the continuity conditions across sub-domains and illustrate it with an example. In order to impose the continuity conditions on the common boundary between two adjacent sub-domains (Ω_i, Ω_j)*, the random boundary collocation points for* Ω_i *and the random boundary collocation points for* Ω_j*, when restricted to their shared boundary, must be identical. We illustrate this point using Figure 1, which shows two adjacent sub-domains in 2D. Two random collocation points* $(N_{bc} = 2)$ *are generated on each boundary of each sub-domain. Note that the collocation points for different sub-domains are generated independently. But we need to make sure that the points for the common face* l *(highlighted in orange in Figure 1) in both sub-domains sharing* l *should be identical.*

This requirement is implemented by the following procedure in our implementation. For each sub-domain we arrange the boundary collocation points in the following order: those random points on the left face, followed by those on the right face, the bottom face, and the top face. We first generate the random boundary points for all sub-domains independently. Then we go through all sub-domains, and the four boundaries (left, right, bottom, top) on each sub-domain, successively. If the boundary being examined is a shared boundary and the ID of the current sub-domain is higher than that of the neighboring sub-domain, then we replace the collocation points for the this boundary of the current sub-domain by those collocation points for the same boundary in the neighboring sub-domain. As shown in Figure 1, one can see that the random collocation points for the left boundary of the second sub-domain will be replaced by (thus identical to) those for the right boundary of the first sub-domain.

Remark 6. *As mentioned previously, the hidden-layer coefficients of the ELM neural network are assigned to uniform random values generated on the interval* $[-R_m, R_m]$ *in this paper. We observe from numerical simulations that the value of the constant* R_m *influences the ELM accuracy; see [15] for a similar observation for low-dimensional PDEs and also [96, 53] for other types of problems with RVFL networks. In the numerical simulations of Section 3 below, we have used* $R_m = R_{m0}$*, where* R_{m0} *is a value determined by the following procedure for a given problem. For a PDE problem of a given dimension* d*, we first fix the number of training parameters in the NN and the number of collocation points to some chosen values. Then we perform preliminary simulations of the given problem using this fixed network setting, and a set of different* R_m *values for generating the random hidden-layer coefficients. We record the errors of the computed solution (when the exact solution is available), or the norm of the residual vector* $\mathbf{R}(\mathbf{\Phi})$ *corresponding to the computed solution (when the exact solution is unavailable), for this set of* R_m *values. We choose the value with the lowest error or the lowest residual norm, and denote it by* R_{m0}*. Then we fix* $R_m = R_{m0}$ *for generating the random hidden-layer coefficients for the subsequent simulations of this PDE problem in a given dimension* d*, while some other simulation parameters (e.g. number of training parameters or collocation points) are varied. We observe that the* R_{m0} *as determined above in general decreases with respect to the problem dimension* d*.*

2.4 Solving High-Dimensional PDEs by Combined ELM and Approximate Theory of Functional Connections (A-TFC)

In this subsection we present an alternative strategy for approximating the solution field $u(x)$, by combining ELM and an approximate variant of the Theory of Functional Connections (TFC). TFC provides a systematic technique for handling linear constraints, in particular the boundary or initial conditions [65, 66, 83, 49]. The number of terms in TFC constrained expression, however, increases exponentially with increasing problem dimension, rendering TFC infeasible for high-dimensional problems. The approximate variant of TFC, termed A-TFC here, avoids the exponential growth in the number of terms of TFC and is suitable for computational PDEs in high dimensions.

2.4.1 TFC and Approximate TFC

Consider the domain $\Omega = [a, b]^d$, where a and b are constants, and a function $u(x)$ defined on Ω satisfying the condition,

$$u|_{\partial\Omega} = C(x), \quad x \in \partial\Omega, \tag{17}$$

where $C(x)$ is a prescribed function on $\partial\Omega$. Then the general form of $u(x)$ is given by [49],

$$u(x) = g(x) - \mathcal{T}_\Omega g(x) + \mathcal{T}_\Omega C(x), \quad x \in \Omega, \tag{18}$$

where g is an arbitrary (free) function on Ω. $\mathcal{T}_\Omega$ is a linear operator satisfying the property that, for any function f defined on $\partial\Omega$, $\mathcal{T}_\Omega f(x)$ is a function defined on Ω and,

$$\mathcal{T}_\Omega f|_{\partial\Omega} = f(x), \quad x \in \partial\Omega. \tag{19}$$

One can verify that $u(x)$ as given by (18) satisfies (17) for an arbitrary g. The expression (18) is called the constrained expression of TFC [49].

For a function f defined on $\partial\Omega$, $\mathcal{T}_\Omega f(x)$ can be constructed as follows. We first use a 2D ($d = 2$) example to illustrate the procedure and then discuss the general d-dimensional case. Suppose $\Omega = [a, b]^2$ and $f(x)$ is a function defined on $\partial\Omega$. Let $c_i^1 \in \{a, b\}$ for $i \in \{1, 2\}$, and define $f_i^{c_i^1}(x) := f(x)|_{x_i = c_i^1}$. Let $c_{1,2}^i \in \{a, b\}$ for $i \in \{1, 2\}$, and define $f_{1,2}^{c_{1,2}^1, c_{1,2}^2}(x) := f(x_1 = c_{1,2}^1, x_2 = c_{1,2}^2)$. We further define

$$\phi_{a,b}^{c_i^1}(x) = \delta_{a,c_i^1}\frac{b - x_i}{b - a} + \delta_{b,c_i^1}\frac{x_i - a}{b - a}, \quad \phi_{a,b}^{c_{1,2}^i}(x) = \delta_{a,c_{1,2}^i}\frac{b - x_i}{b - a} + \delta_{b,c_{1,2}^i}\frac{x_i - a}{b - a}, \quad i \in \{1, 2\}. \tag{20}$$

Then

$$\mathcal{T}_\Omega f(x) = T_\Omega^1 f(x) - T_\Omega^2 f(x), \tag{21}$$

where

$$\begin{cases} \mathcal{T}_\Omega^1 f(x) = \displaystyle\sum_{i=1}^{2} \sum_{c_i^1 \in \{a,b\}} \phi_{a,b}^{c_i^1}(x) f_i^{c_i^1}(x) = \sum_{i=1}^{2} \left[\frac{b - x_i}{b - a} f(x)|_{x_i = a} + \frac{x_i - a}{b - a} f(x)|_{x_i = b} \right], \\ \mathcal{T}_\Omega^2 f(x) = \displaystyle\sum_{c_{1,2}^1, c_{1,2}^2 \in \{a,b\}} \phi_{a,b}^{c_{1,2}^1}(x) \phi_{a,b}^{c_{1,2}^2}(x) f_{1,2}^{c_{1,2}^1, c_{1,2}^2}(x) \\ \qquad = \dfrac{(b - x_1)(b - x_2)}{(b - a)^2} f(a, a) + \dfrac{(b - x_1)(x_2 - a)}{(b - a)^2} f(a, b) + \dfrac{(x_1 - a)(b - x_2)}{(b - a)^2} f(b, a) \\ \qquad\quad + \dfrac{(x_1 - a)(x_2 - a)}{(b - a)^2} f(b, b). \end{cases} \tag{22}$$

For the general case, one can construct $\mathcal{T}_\Omega f$ in a similar fashion. The domain $\Omega = [a, b]^d$ is a d-dimensional hypercube (referred to as d-cube hereafter). We need to consider all j-cubes contained on the boundary $\partial\Omega$, for $j = 0, \cdots, d - 1$. To this end, we define the following notations. For $j = 1, \ldots, d$, we define $\mathcal{E}_j$ as the

collection of all j tuples $(k_1, \ldots, k_j)$ with $1 \leq k_1 < k_2 < \cdots < k_j \leq d$. The cardinality of $\mathcal{E}_j$ is $\begin{bmatrix} d \\ j \end{bmatrix}$. Hence, there are $2^j \begin{bmatrix} d \\ j \end{bmatrix}$ $(d-j)$-cubes on $\partial\Omega$. We define

$$\begin{cases} f_{\mathbf{p}}^{c_{\mathbf{p}}^1,\ldots,c_{\mathbf{p}}^j}(x) = f(x)|_{(x_{p_1}=c_{\mathbf{p}}^1, x_{p_2}=c_{\mathbf{p}}^2,\ldots,x_{p_j}=c_{\mathbf{p}}^j)}, \quad \mathbf{p} = (p_1,\ldots,p_j) \in \mathcal{E}_j; \\ \phi_{a,b}^{c_{\mathbf{p}}^i}(x) = \delta_{a,c_{\mathbf{p}}^i} \dfrac{b - x_{p_i}}{b-a} + \delta_{b,c_{\mathbf{p}}^i} \dfrac{x_{p_i} - a}{b-a}, \quad \mathbf{p} \in \mathcal{E}_j, \quad i \in [j], \end{cases} \tag{23}$$

where $c_{\mathbf{p}}^i \in \{a, b\}$ for $1 \leqslant i \leqslant j$. Then $\mathcal{T}_\Omega f(x)$ is given by

$$\mathcal{T}_\Omega f(x) = \sum_{i=1}^{d} (-1)^{i-1} \mathcal{T}_\Omega^i f(x), \tag{24}$$

where

$$\mathcal{T}_\Omega^i f(x) = \sum_{\mathbf{p} \in \mathcal{E}_i} \sum_{c_{\mathbf{p}}^1,\ldots,c_{\mathbf{p}}^i \in \{a,b\}} \phi_{a,b}^{c_{\mathbf{p}}^1}(x) \cdots \phi_{a,b}^{c_{\mathbf{p}}^i}(x) f_{\mathbf{p}}^{c_{\mathbf{p}}^1,\ldots,c_{\mathbf{p}}^i}(x), \quad \forall i \in [d]. \tag{25}$$

While the constrained expression (18) satisfies the condition (17) exactly, the number of terms contained therein grows exponentially with respect to the dimension d. The expression (25) contains $2^i \begin{bmatrix} d \\ i \end{bmatrix}$ terms, giving rise to a total number of $(3^d - 1)$ terms in (24). Therefore, the constrained expression (18) contains $\mathcal{O}(3^d)$ terms in d dimensions, rendering TFC infeasible for high-dimensional PDE problems.

To devise a TFC-like approximation of the solution field suitable for high-dimensional problems, we notice that the expression (24) represents a hierarchical decomposition of $\mathcal{T}_\Omega f(x)$ in some sense, in which $\mathcal{T}_\Omega^i f(x)$ represents the contributions of $f(x)$ from the $(d-i)$-dimensional hyperplanes on the boundary $\partial\Omega$. Taking $d = 3$ as an example, one can note that $\mathcal{T}_\Omega^1 f$, $\mathcal{T}_\Omega^2 f$ and $\mathcal{T}_\Omega^3 f$ represent the contributions of $f(x)$ from the faces, the edges, and the vertices of the cube $\Omega = [a, b]^3$.

This observation inspires the following strategy for approximating $\mathcal{T}_\Omega f(x)$. We can truncate the expression (24) by keeping only the leading terms, in a spirit analogous to the truncation in Taylor expansion. Specifically, by choosing a number $k \in [d-1]$ as the cut-off value, we retain all the terms $\mathcal{T}_\Omega^j f$ with $1 \leqslant j \leqslant k$ in (24) and discard the rest of the terms. In the current paper we choose $k = 1$ for simplicity. In other words, for a function $f(x)$ defined on $\partial\Omega$, we define $\mathcal{A}_\Omega f(x)$ by

$$\mathcal{A}_\Omega f(x) := \mathcal{T}_\Omega^1 f(x), \quad x \in \Omega. \tag{26}$$

Then we approximate the function $u(x)$ satisfying the condition (17) by,

$$u(x) = g(x) - \mathcal{A}_\Omega g(x) + \mathcal{A}_\Omega C(x), \tag{27}$$

where $g(x)$ is a function to be determined. We refer to the approximation (27) as the approximate TFC (or A-TFC) constrained expression.

The terms $\mathcal{A}_\Omega g$ and $\mathcal{A}_\Omega C$ in the A-TFC expression (27) both contain $2d$ terms. The computation of (27) is thus feasible for large d, and A-TFC avoids the exponential growth in the number of terms of the TFC constrained expression (18). However, there is a trade-off with A-TFC. Specifically, the A-TFC expression (27) for $u(x)$ does not satisfy the boundary condition (17) unconditionally for an arbitrary function $g(x)$, because $\mathcal{A}_\Omega f$ is an approximation of $\mathcal{T}_\Omega f$. In general, $g(x)$ needs to satisfy a certain condition on $\partial\Omega$ in order for $u(x)$ to satisfy (17).

We first illustrate this point using the 2D case. Suppose $\Omega = [a, b]^2$, and we substitute the expression (27) into (17). On the boundary $x_2 = a$, we have

$$\begin{aligned} C(x_1, a) &= \mathcal{A}C(x_1, a) + g(x_1, a) - \mathcal{A}g(x_1, a) \\ &= C(x_1, a) + [C(a, a) - g(a, a)] \frac{b - x_1}{b-a} + [C(b, a) - g(b, a)] \frac{x_1 - a}{b-a}. \end{aligned} \tag{28}$$

This leads to the following condition for g,

$$[C(a,a)-g(a,a)]\frac{b-x_1}{b-a}+[C(b,a)-g(b,a)]\frac{x_1-a}{b-a}=0,\quad \forall x_1\in[a,b]. \tag{29}$$

Similarly, by considering the other boundaries we attain the following conditions,

$$\begin{cases}[C(a,b)-g(a,b)]\frac{b-x_1}{b-a}+[C(b,b)-g(b,b)]\frac{x_1-a}{b-a}=0, & \forall x_1\in[a,b];\\ [C(a,a)-g(a,a)]\frac{b-x_2}{b-a}+[C(a,b)-g(a,b)]\frac{x_2-a}{b-a}=0, & \forall x_2\in[a,b];\\ [C(b,a)-g(b,a)]\frac{b-x_2}{b-a}+[C(b,b)-g(b,b)]\frac{x_2-a}{b-a}=0, & \forall x_2\in[a,b].\end{cases} \tag{30}$$

In this 2D case, it is straightforward to choose $g(a,a)=C(a,a)$, $g(a,b)=C(a,b)$, $g(b,a)=C(b,a)$ and $g(b,b)=C(b,b)$ as the boundary conditions for g. However, in high-dimensional cases the conditions given by (29) and (30) are easier to implement with randomly chosen collocation points.

For the general d-dimensional case, we again use the notation in (23). The conditions for g are then given by,

$$\sum_{j\neq i}\left[\left(C^{a,a}_{i,j}-g^{a,a}_{i,j}\right)\frac{b-x_j}{b-a}+\left(C^{a,b}_{i,j}-g^{a,b}_{i,j}\right)\frac{x_j-a}{b-a}\right]=0,\quad \text{on } x_i=a,\quad \forall i\in[d]; \tag{31a}$$

$$\sum_{j\neq i}\left[\left(C^{b,a}_{i,j}-g^{b,a}_{i,j}\right)\frac{b-x_j}{b-a}+\left(C^{b,b}_{i,j}-g^{b,b}_{i,j}\right)\frac{x_j-a}{b-a}\right]=0,\quad \text{on } x_i=b,\quad \forall i\in[d]. \tag{31b}$$

We define $\tilde{\mathcal{B}}_\Omega$ as

$$\tilde{\mathcal{B}}_\Omega f(x)=\begin{bmatrix}\vdots\\ \sum_{j\neq i}\left(f^{a,a}_{i,j}\frac{b-x_j}{b-a}+f^{a,b}_{i,j}\frac{x_j-a}{b-a}\right)\\ \sum_{j\neq i}\left(f^{b,a}_{i,j}\frac{b-x_j}{b-a}+f^{b,b}_{i,j}\frac{x_j-a}{b-a}\right)\\ \vdots\end{bmatrix},\quad x\in\partial\Omega. \tag{32}$$

Then one can rewrite (31) as

$$\tilde{\mathcal{B}}_\Omega g(x)=s_\Omega(x),\quad x\in\partial\Omega, \tag{33}$$

where $s_\Omega(x):=\tilde{\mathcal{B}}_\Omega C(x)$ denotes the boundary data. This is the boundary condition for $g(x)$ with A-TFC.

We can make the following observation from the above discussions. With A-TFC, while the constrained expression does not satisfy the condition (17) exactly for an arbitrary g, the condition that g needs to satisfy generally involves functions of a simpler form than the original condition. Specifically, there are $2(d-1)$ terms in the condition for g on each boundary, and each term involves the product of a linear function and the g evaluated on a $(d-2)$-dimensional hyperplane of the boundary. Therefore, A-TFC can simplify the functional forms involved in the condition in some sense. Let us again use 2D as an example to illustrate this point. Assume a boundary distribution $C(x_1,a)=\sin(x_1)$ on the boundary $x_2=a$. Then, without A-TFC, the boundary condition for $u(x)$ on $x_2=a$ is given by,

$$u(x_1,a)=\sin(x_1),\quad \forall x_1\in[a,b].$$

In contrast, with A-TFC, the condition for g on $x_2=a$ is reduced to (see (29)),

$$[\sin(a)-g(a,a)]\frac{b-x_1}{b-a}+[\sin(b)-g(b,a)]\frac{x_1-a}{b-a}=0,\quad \forall x_1\in[a,b].$$

The linear function involved in the condition for g with A-TFC is obviously simpler than that of the original condition without A-TFC.

2.4.2 A-TFC Embedded ELM

Let us now consider how to combine ELM and A-TFC for solving the system (1). In this sub-section we will assume that $\mathcal{B} = \mathbf{I}$ (identity operator) in (1b), i.e. the problem has Dirichlet boundary conditions.

Based on A-TFC, we perform the following transformation,

$$u(x) = g(x) - \mathcal{A}_\Omega g(x) + \mathcal{A}_\Omega H(x), \quad x \in \Omega, \tag{34}$$

where $g(x)$ is an unknown function, $\mathcal{A}_\Omega$ is defined in (26), and H is the boundary distribution in (1b). The system (1) is accordingly transformed into,

$$\mathcal{L}g(x) - \mathcal{L}\mathcal{A}_\Omega g(x) + \mu\mathcal{N}(\mathcal{A}_\Omega H(x) + g(x) - \mathcal{A}_\Omega g(x)) = Q(x) - \mathcal{L}\mathcal{A}_\Omega H(x) = Q_1(x), \quad x \in \Omega, \tag{35a}$$

$$\tilde{\mathcal{B}}_\Omega g(x) = \tilde{\mathcal{B}}_\Omega H(x) = S_\Omega(x), \quad x \in \partial\Omega, \tag{35b}$$

where $\tilde{\mathcal{B}}_\Omega$ is defined in (32). $g(x)$ is the field function to be solved for in (35).

We represent $g(x)$ by an ELM-type randomized neural network, following the settings as outlined in Section 2.3. In particular, the input layer of the network contains d nodes (representing x), and the output layer contains a single node (representing $g(x)$) with zero bias and no activation function. The hidden-layer coefficients are set to uniform random values from the interval $[-R_m, R_m]$. Let N_g denote the number of nodes in the last hidden layer of the neural network, and $V_j(x)$ $(1 \leqslant j \leqslant N_g)$ denote the output fields of the last hidden layer. Then the network logic gives rise to,

$$g(x) = \sum_{j=1}^{N_g} \phi_j V_j(x) = \mathbf{V}(x)\mathbf{\Phi}, \tag{36}$$

where $\mathbf{V}(x) = (V_1(x), \ldots, V_{N_g}(x))$, ϕ_j $(1 \leqslant j \leqslant N_g)$ are the output-layer coefficients (training parameters), and $\mathbf{\Phi} = (\phi_1, \ldots, \phi_{N_g})^T$.

We determine the training parameters $\mathbf{\Phi}$ in (36) in a fashion similar to in Section 2.3. We again employ random collocation points inside the domain and on the domain boundaries, and let N_{in} and N_{bc} denote the number of interior collocation points and the number of boundary collocation points on each hyperface, respectively. Enforcing the residual function of the system (35) to be zero on these collocation points leads to the following algebraic system,

$$\mathbf{R}(\mathbf{\Phi}) = \begin{bmatrix} \mathcal{L}(\mathbf{V}(\mathbf{x}) - \mathcal{A}_\Omega \mathbf{V}(\mathbf{x}))\mathbf{\Phi} + \mu\mathcal{N}(\mathcal{A}_\Omega H(\mathbf{x}) + \mathbf{V}(\mathbf{x})\mathbf{\Phi} - \mathcal{A}_\Omega \mathbf{V}(\mathbf{x})\mathbf{\Phi}) - Q_1(\mathbf{x}) \\ \tilde{\mathcal{B}}_\Omega \mathbf{V}(\mathbf{y})\mathbf{\Phi} - S_\Omega(\mathbf{y}) \end{bmatrix} = \mathbf{0}, \tag{37}$$

where $\mathbf{x}$ and $\mathbf{y}$ are defined in (11), and $\mathbf{R}(\mathbf{\Phi}) \in \mathbb{R}^{N_a}$ denotes the residual vector of the system (35) on the collocation points, where $N_a = N_{\text{in}} + 2N_{\text{bc}}^{\text{tot}} = N_{\text{in}} + 4dN_{\text{bc}}$.

We seek a least squares solution to the system (37), and solve this system for $\mathbf{\Phi}$ by the linear least squares method, if $\mu = 0$, and by the nonlinear least squares method with perturbations (NLLSQ-perturb) [15], if $\mu \neq 0$. When $\mu = 0$, the system (37) is reduced to,

$$\begin{bmatrix} \mathcal{L}(\mathbf{V}(\mathbf{x}) - \mathcal{A}_\Omega \mathbf{V}(\mathbf{x})) \\ \tilde{\mathcal{B}}_\Omega \mathbf{V}(\mathbf{y}) \end{bmatrix} \mathbf{\Phi} = \begin{bmatrix} Q_1(\mathbf{x}) \\ S_\Omega(\mathbf{y}) \end{bmatrix}, \tag{38}$$

which is linear, and $\mathbf{\Phi}$ can be computed by the linear least squares method. When $\mu \neq 0$, the system (37) is nonlinear. The Jacobian matrix of this system is given by,

$$\mathbf{J}(\mathbf{\Phi}) = \frac{\partial \mathbf{R}}{\partial \mathbf{\Phi}} = \begin{bmatrix} \mathcal{L}(\mathbf{V}(\mathbf{x}) - \mathcal{A}_\Omega \mathbf{V}(\mathbf{x})) + \mu\mathcal{N}'(\mathcal{A}_\Omega H(\mathbf{x}) + \mathbf{V}(\mathbf{x})\mathbf{\Phi} - \mathcal{A}_\Omega \mathbf{V}(\mathbf{x})\mathbf{\Phi})(\mathbf{V}(\mathbf{x}) - \mathcal{A}_\Omega \mathbf{V}(\mathbf{x})) \\ \tilde{\mathcal{B}}_\Omega \mathbf{V}(\mathbf{y}) \end{bmatrix}, \tag{39}$$

which is needed by the NLLSQ-perturb method [15] for solving (37).

Upon solving the system (37) by the linear or nonlinear least squares method, we set the output-layer coefficients of the neural network by the least squares solution $\mathbf{\Phi}$. Afterwards, a forward evaluation of the neural network provides $g(x)$, and the solution field to system (1) is computed by (34).

3 Numerical Examples

In this section we test the performance of the proposed methods using several high-dimensional PDEs. These include the linear and nonlinear Poisson equations, which are time-independent, and the heat, Korteweg-de Veries (KdV), and the advection diffusion equations, which are time-dependent. For each problem we investigate the error convergence with respect to the number of training parameters and number of collocation points for a range of problem dimensions.

A single hidden layer has been employed in the neural network with both the ELM and the ELM/A-TFC methods for all test problems. In subsequent discussions we denote the neural network architecture by the following vector (or list) of positive integers, referred to as the architectural vector henceforth,

$$\mathbf{M}_{\text{arch}} = [m_{\text{in}}, M, m_{\text{out}}] \tag{40}$$

where m_{in}, M and m_{out} are the number of nodes in the input layer, the hidden layer, and the output layer, respectively. $m_{\text{out}} = 1$ in all tests of this section, and M equals the number of training parameters of ELM. The hidden-layer coefficients are assigned to uniform random values generated on the interval $[-R_m, R_m]$ with $R_m = R_{m0}$ in the simulations below, where R_{m0} is determined by the procedure discussed in Remark 6. The tanh activation function has been employed in the hidden layer for all the numerical tests of this section.

In all numerical experiments, after the NN with a particular setting (given number of training parameters, and given number of training collocation points) is trained, we compute the errors of the network solution as follows. We generate a set of test points on the d-dimensional domain Ω: $N_{\text{in}}^{(v)}$ random points on the interior of Ω, and $N_{\text{bc}}^{(v)}$ random points on each boundary of Ω ($(d-1)$-dimensional hypercube). There is a total of $N_c^{(v)} = N_{\text{in}}^{(v)} + 2dN_{\text{bc}}^{(v)}$ random test points. They are different from the random collocation points used in the network training, and the number is much larger than that of the latter. We evaluate the trained NN on these test points to obtain the NN solution data, and evaluate the exact solution to the problem on the same test points. We compare the data of the NN solution and the exact solution on these test points to compute the maximum error (e_{l^∞}) and the root-mean-squares (rms) error (e_{l^2}) as follows,

$$e_{l^\infty} = \max\{\ |u(\mathbf{x}_i) - u_{ex}(\mathbf{x}_i)|\ \}_{i=1}^{N_c^{(v)}}, \quad e_{l^2} = \sqrt{\frac{1}{N_c^{(v)}} \sum_{i=1}^{N_c^{(v)}} [u(\mathbf{x}_i) - u_{ex}(\mathbf{x}_i)]^2}, \tag{41}$$

where $\mathbf{x}_i$ ($1 \leqslant i \leqslant N_c^{(v)}$) denote the test points, and $u(\mathbf{x}_i)$ and $u_{ex}(\mathbf{x}_i)$ denote the NN solution and the exact solution, respectively. We refer to the computed e_{l^∞} and e_{l^2} as the errors associated with the given NN setting that is used for training the network. In all the numerical simulations of this section we employ a fixed $N_{\text{bc}}^{(v)} = 100$ and $N_{\text{in}}^{(v)} = 7000$ for computing the e_{l^∞} and e_{l^2} errors.

In the numerical tests below, d refers to the dimension of the spatial domain. For time-dependent problems, the time variable t is not included in d when we talk about the problem in d dimensions. In other words, for a d-dimensional time-dependent problem, the input layer of the neural network would contain $m_{\text{in}} = d + 1$ nodes (space and time). To make the reported results in this section exactly re-producible, we have set the seed to a fixed value 1 in the random number generators of the Numpy and Tensorflow libraries for all the numerical examples. The test results with the ELM method are presented in Section 3.1 first, and those obtained with the combined ELM/A-TFC method are discussed in Section 3.2. We then compare the current ELM method with the PINN method [78] for selected test problems in Section 3.3.

3.1 Numerical Tests with the ELM Method

3.1.1 Poisson Equation

We consider the Poisson equation on the domain $\Omega = [-1, 1]^d$,

$$-\Delta u = f(x), \quad x \in \Omega, \tag{42a}$$

$$u = h(x), \quad x \in \partial\Omega, \tag{42b}$$

R_m	0.01	0.05	0.1	0.15	0.25	0.5
e_{l^∞}	1.76E-6	1.30E-8	1.04E-7	1.23E-6	4.44E-5	8.33E-4
e_{l^2}	1.96E-7	6.68E-10	6.91E-9	8.48E-8	2.28E-6	9.03E-5

Table 1: Poisson equation: Determining the R_{m0} based on the procedure of Remark 6 for $d = 5$. NN architecture: $[d, 2000, 1]$; $(N_{\rm bc}, N_{\rm in}) = (100, 1000)$.

d	5	7	15
R_{m0}	0.05	0.05	0.001

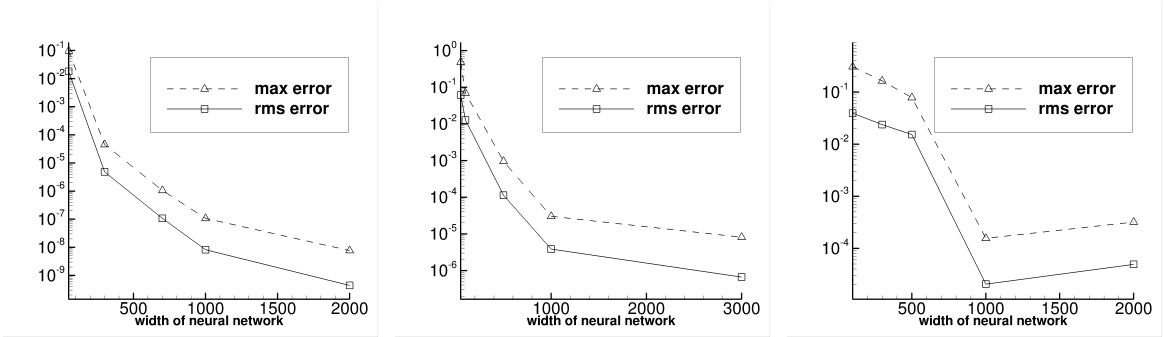
Table 2: Poisson equation: R_{m0} determined by the procedure from Remark 6 for several problem dimensions.

where $\Delta = \sum_{i=1}^d \frac{\partial^2}{\partial x_i^2}$, $h(x) = \left(\frac{1}{d}\sum_{i=1}^d x_i\right)^2 + \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right)$, and $f(x) = \frac{1}{d}\left(\sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right) - 2\right)$. The exact solution to this system is $u(x) = \left(\frac{1}{d}\sum_{i=1}^d x_i\right)^2 + \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right)$.

We solve the system (42) by the ELM method with an NN architecture $\mathbf{M}_{\rm arch} = [d, M, 1]$, where the d input nodes represent x, the single output node represents $u(x)$, and the hidden-layer width M (i.e. number of training parameters) is varied systematically. The neural network is trained by the algorithm from Section 2.3 on a set of random collocation points, consisting of $N_{\rm in}$ points from the interior of Ω and $N_{\rm bc}$ points on each of the $2d$ boundaries of $\partial\Omega$, where $(N_{\rm bc}, N_{\rm in})$ are varied in the tests. For the network training, the input data consists of the coordinates of all the collocation points. After the NN is trained for each case, as discussed previously, the maximum and rms errors of the NN solution are computed on another set of random test points, characterized by $(N_{\rm bc}^{(v)}, N_{\rm in}^{(v)}) = (100, 7000)$ for the test points on each boundary and in the interior of Ω. In addition, on selected 2D cross sections of Ω, such as the x_i-x_j plane $(1 \leqslant i < j \leqslant d)$, we have evaluated the network solution on a set of regular $Q^{(v)} \times Q^{(v)}$ grid points ($Q^{(v)} = 800$), and compared with the exact solution on the same set of points to study the point-wise errors of the NN solution.

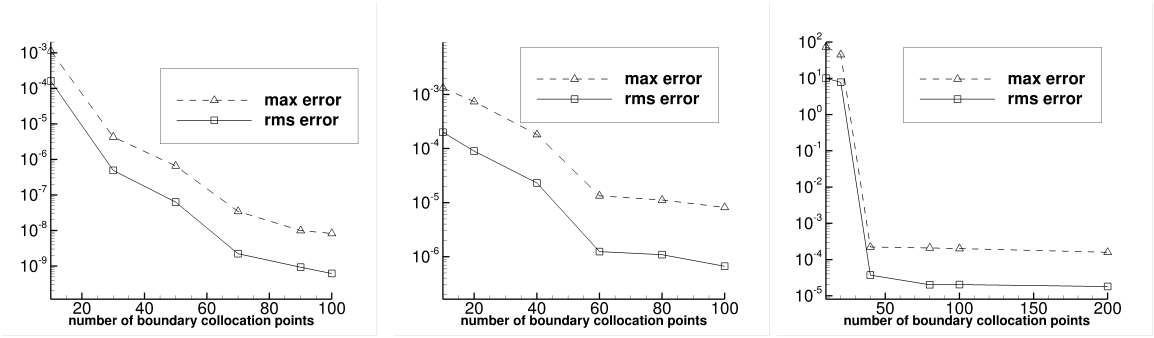
We first determine the R_{m0} based on the procedure from Remark 6 for generating the random hidden-layer coefficients. Table 1 shows the maximum and rms errors of the ELM solution for dimension $d = 5$, corresponding to several R_m values for generating the random hidden-layer coefficients. These are obtained using a network architecture $[d, 2000, 1]$ and the collocation points $(N_{\rm bc}, N_{\rm in}) = (100, 1000)$. The ELM method produces accurate results in a range of R_m values around $R_m \approx 0.05$, thus leading to $R_{m0} = 0.05$ for $d = 5$. Table 2 lists the R_{m0} for several problem dimensions with the Poisson equation, which are obtained with the same network architecture and the same number of collocation points as in Table 1. We observe that R_{m0} tends to decrease as the dimension d increases. This seems to be a characteristic common to all the test problems in this work. In the subsequent simulations, we set $R_m = R_{m0}$ in ELM for generating the hidden-layer coefficients, while the other simulation parameters are varied.

Figure 2 illustrates the effect of the number of training parameters on the ELM accuracy. It shows the



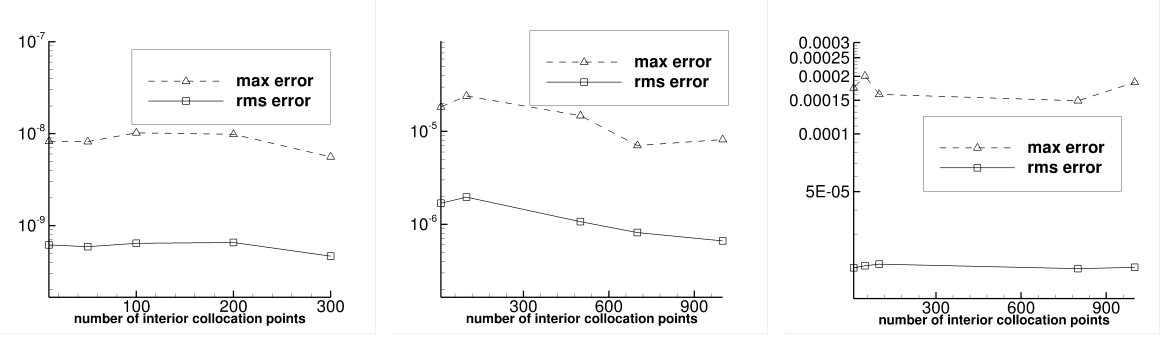
(a) d=5 (b) d=7 (c) d=15

Figure 2: Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of training parameters (M) for dimensions (a) $d = 5$, (b) $d = 7$, (c) $d = 15$. $(N_{\rm bc}, N_{\rm in}) = (100, 2000)$ in (a), $(100, 1000)$ in (b), and $(120, 2000)$ in (c).



(a) d=5 (b) d=7 (c) d=15

Figure 3: Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of boundary collocation points ($N_{\rm bc}$) for dimensions (a) $d = 5$, (b) $d = 7$, and (c) $d = 15$. NN architecture: $[d, 2000, 1]$ in (a,b,c); $N_{\rm in} = 500$ in (a), 1000 in (b), and 50 in (c).



(a) d=5 (b) d=7 (c) d=15

Figure 4: Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of interior collocation points ($N_{\rm in}$) for dimensions (a) $d = 5$, (b) $d = 7$, and (c) $d = 15$. NN architecture: $[d, 2000, 1]$ in (a,b,c); $N_{\rm bc} = 100$ in (a,b,c).

maximum and rms errors of ELM versus the number of training parameters (M) for the Poisson equation in $d = 5$, 7 and 15 dimensions. The other crucial simulation parameters are listed in the figure caption. It is observed that the errors decrease quasi-exponentially with increasing number of training parameters (when $M \lesssim 1000$), but appears to stagnate at a certain level as M increases further. The stagnant error level tends to be larger with a higher dimension. For example, the e_{l^2} error level is on the order of 10^{-10}, 10^{-7} and 10^{-5} for the dimensions $d = 5$, 7 and 15, respectively.

Figures 3 and 4 illustrate the effect of the training collocation points on the ELM accuracy. Figure 3 shows the ELM errors (e_{l^∞} and e_{l^2}) as a function of the number of collocation points on each boundary ($N_{\rm bc}$) for dimensions $d = 5$, 7 and 15. Figure 4 shows the ELM errors as a function of the number of interior collocation points ($N_{\rm in}$). The other simulation parameters are fixed in the tests and their values are provided in the captions of these figures. Increasing the number of boundary collocation points ($N_{\rm bc}$) improves the ELM accuracy significantly. The ELM maximum/rms errors decrease approximately exponentially with increasing $N_{\rm bc}$ for $d = 5$, and also for $d = 7$ when $N_{\rm bc} \lesssim 60$ or $d = 15$ when $N_{\rm bc} \lesssim 40$. The errors stagnate when $N_{\rm bc}$ increases beyond around 60 for $d = 7$ and beyond 40 for $d = 15$. On the other hand, varying the number of interior collocation points $N_{\rm in}$ appears to have little effect on the ELM accuracy for all three dimensions, which is evident from Figure 4. With increasing problem dimension, the surface of the hypercube (and hence the boundary collocation points) becomes more dominant, while the interior (hence the interior collocation points) becomes less important. Therefore, a small number of interior collocation points will typically suffice in higher dimensions.

In all the studies so far, the e_{l^∞} and e_{l^2} errors are evaluated on a finite set of random test points from the

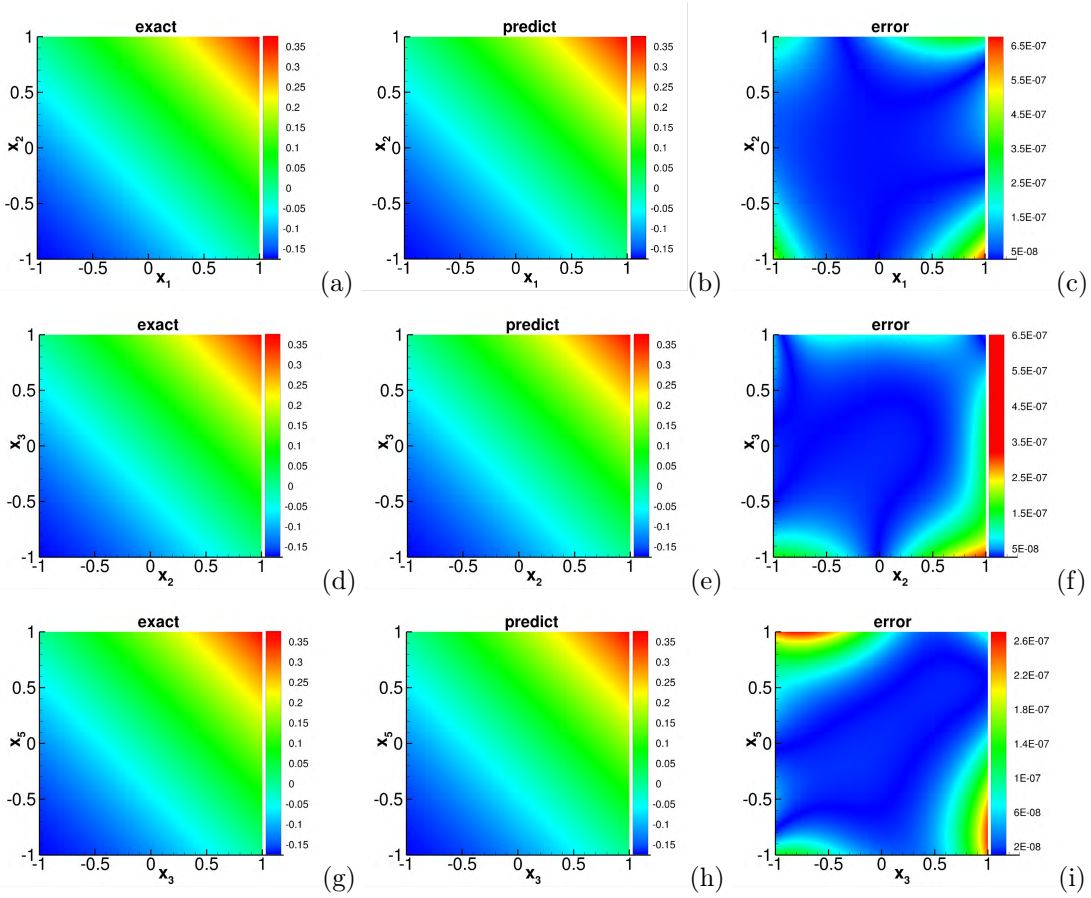


Figure 5: Poisson equation ($d = 7$): Distributions of the exact solution (left column), the ELM solution (middle column), and the point-wise absolute error of ELM (right column), in selected cross sections of the domain. Top row, x_1-x_2 plane; Middle row, x_2-x_3 plane; Bottom row, x_3-x_5 plane. NN architecture: $[d, 2000, 1]$; $(N_{\rm bc}, N_{\rm in}) = (100, 1000)$.

domain, characterized by $(N_{\rm bc}^{(v)}, N_{\rm in}^{(v)}) = (100, 7000)$. We have observed the error levels on the order of 10^{-10} to 10^{-5} for the problem dimensions from $d = 5$ to 15. We would like to consider the following question. Are these error levels representative of the ELM solution error on the entire domain Ω?

To answer this question, ideally one would generate a regular set of grid points on Ω, with a sufficiently large number of grid points in each direction, and then evaluate and visualize the point-wise errors of the ELM solution on these grid points. This is feasible for low dimensions, but immediately becomes impractical when the dimension d increases to even a moderate value. On the other hand, we note that it is possible to extract/compute the ELM solution error on certain low-dimensional hyper-planes (e.g. 2D cross sections) in high dimensions. By looking into the point-wise error distributions in selected cross sections, one can gain a general sense of the representative error levels in the domain.

Figure 5 is an illustration of the ELM error for the Poisson equation in dimension $d = 7$, using cross sections. It shows distributions of the exact solution, the ELM solution, and the point-wise absolute error of ELM, on three cross sections of the domain (x_1-x_2 plane, x_2-x_3 plane, and x_3-x_5 plane). For a selected x_i-x_j plane, the other coordinates of this plane has been set to zero, $x_k = 0$ for $k \neq i, j$ (i.e. the middle of the domain). For each cross section, the ELM solution, the exact solution, and the ELM error are evaluated on a uniform set of $Q^{(v)} \times Q^{(v)}$ ($Q^{(v)} = 800$) grid points. The other simulation parameters are listed in the caption of this figure. One can observe that the point-wise error levels as shown in these cross sections are comparable to (or consistent with) those observed in the convergence studies for $d = 7$. This suggests that the e_{l^∞} and e_{l^2} errors computed on the random set of test points with $(N_{\rm bc}^{(v)}, N_{\rm in}^{(v)}) = (100, 7000)$ indeed seems to reflect well the ELM error on the domain Ω.

R_m	0.01	0.05	0.1	0.5	1	2
e_{l^∞}	5.44E-6	4.10E-9	2.59E-10	1.65E-8	5.92E-6	4.59E-4
e_{l^2}	9.07E-7	7.57E-10	3.16E-11	3.79E-9	1.25E-6	9.01E-5

Table 3: Nonlinear Poisson equation: Determining R_{m0} by the procedure from Remark 6 for $d = 3$. NN architecture: $[d, 1000, 1]$; $(N_{\rm bc}, N_{\rm in}) = (100, 100)$.

d	3	5	9
R_{m0}	0.1	0.05	0.001

Table 4: Nonlinear Poisson equation: R_{m0} determined by the procedure from Remark 6 for several dimensions.

3.1.2 Nonlinear Poisson Equation

We consider the domain $\Omega = [-1, 1]^d$ and the following problem on Ω,

$$-\nabla \cdot (a(u)\nabla u) = f(x), \quad x \in \Omega, \tag{43a}$$

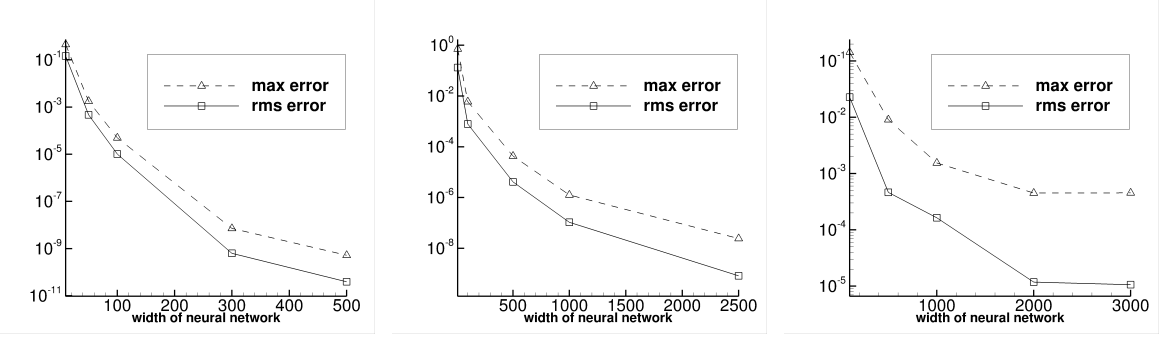
$$u = g(x), \quad x \in \partial\Omega, \tag{43b}$$

where $a(u) = u^2 - u$, $g(x) = \exp\left(-\frac{1}{d}\sum_{i=1}^d x_i\right)$, and $f(x) = \frac{1}{d}\left[-3\exp\left(-\frac{3}{d}\sum_{i=1}^d x_i\right) + 2\exp\left(-\frac{2}{d}\sum_{i=1}^d x_i\right)\right]$. This system has an exact solution $u(x) = \exp\left(-\frac{1}{d}\sum_{i=1}^d x_i\right)$.

We employ the same notations as in Section 3.1.1. The simulation parameters include the network architecture $\mathbf{M}_{\rm arch} = [d, M, 1]$, the random collocation points characterized by $(N_{\rm bc}, N_{\rm in})$, the random test points characterized by $(N_{\rm bc}^{(v)}, N_{\rm in}^{(v)}) = (100, 7000)$, the set of $Q^{(v)} \times Q^{(v)}$ uniform grid points with $Q^{(v)} = 800$ on selected cross sections of the domain for evaluating the ELM and exact solutions, and $R_m = R_{m0}$ for generating the random hidden-layer coefficients in the ELM neural network.

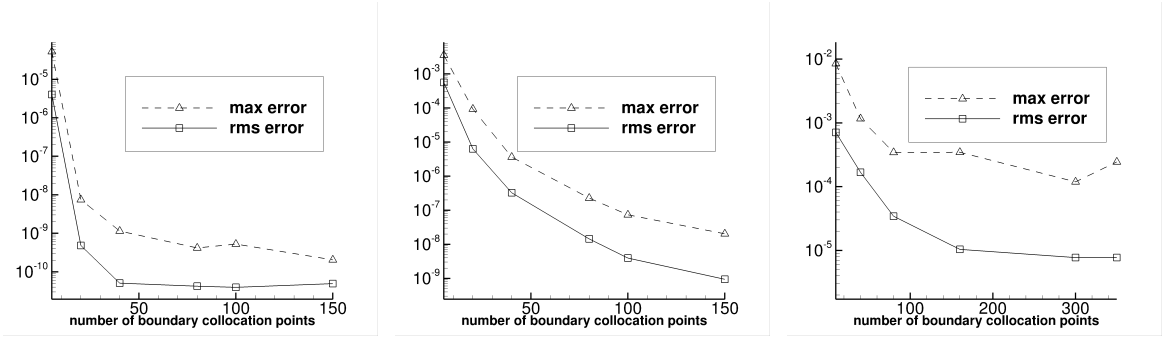
We first employ a fixed network architecture $[d, 1000, 1]$ and a set of collocation points characterized by $(N_{\rm bc}, N_{\rm in}) = (100, 100)$ to determine R_{m0} by the procedure from Remark 6. Table 3 lists the e_{l^∞} and e_{l^2} errors corresponding to several R_m values for generating the random hidden-layer coefficients in ELM for $d = 3$, which leads to $R_{m0} \approx 0.1$. Table 4 lists the R_{m0} values determined by this procedure for several dimensions ranging from $d = 3$ to $d = 9$, using the same simulation parameters (network architecture, collocation points) as in Table 3. One can again observe that R_{m0} decreases with increasing d. We employ $R_m = R_{m0}$ when generating the random hidden-layer coefficients in ELM in the subsequent tests.

Figure 6 illustrates the convergence behavior of the ELM errors with respect to the number of training parameters. It shows the e_{l^∞} and e_{l^2} errors as a function of the number of training parameters (M) for three problem dimensions ($d = 3$, 5, 9). The number of training collocation points is fixed, and their values are



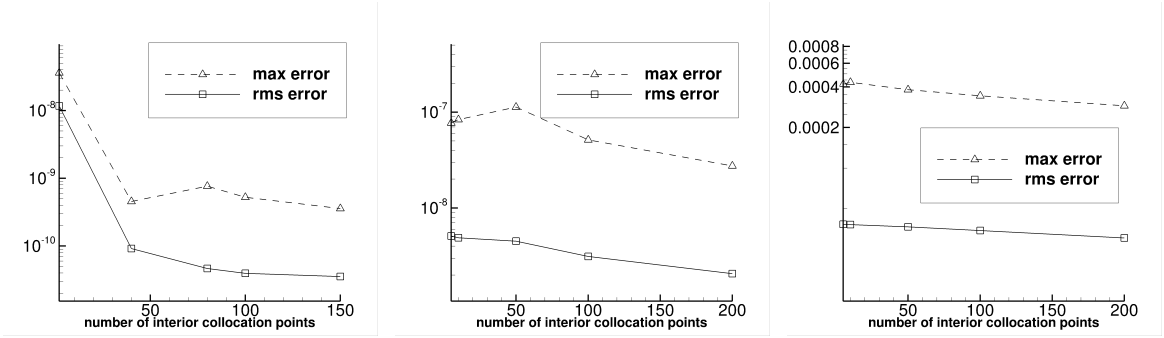
(a) d=3 (b) d=5 (c) d=9

Figure 6: Nonlinear Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of training parameters (M) for dimensions (a) $d = 3$, (b) $d = 5$, and (c) $d = 9$. NN architecture: $[d, M, 1]$ in (a,b,c); $(N_{\rm bc}, N_{\rm in}) = (100, 100)$ in (a,b) and (100,220) in (c).



(a) d=3 (b) d=5 (c) d=9

Figure 7: Nonlinear Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of collocation points on each boundary ($N_{\rm bc}$). NN architecture: $[d, M, 1]$ with $M = 500$ in (a), $M = 2500$ in (b), and $M = 3000$ in (c). $N_{\rm in} = 100$ in (a,b,c).



(a) d=3 (b) d=5 (c) d=9

Figure 8: Nonlinear Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of interior collocation points ($N_{\rm in}$). NN architecture: $[d, M, 1]$ with $M = 500$ in (a), $M = 2500$ in (b), and $M = 3000$ in (c). $N_{\rm bc} = 100$ in (a,b) and 80 in (c).

provided in the figure caption. The ELM errors decrease dramatically (quasi-exponentially) with increasing M initially, but gradually plateau when M becomes large. The e_{l^2} error reaches a level around 10^{-11} for $d = 3$, around 10^{-9} for $d = 5$, and around 10^{-5} for $d = 9$ in the range of parameters tested here. The nonlinear least squares (Gauss-Newton) method typically takes several dozen iterations to converge.

The convergence of the ELM errors with respect to the number of collocation points is illustrated by Figures 7 and 8, which show the e_{l^∞} and e_{l^2} errors as a function of the boundary collocation points ($N_{\rm bc}$) and the interior collocation points ($N_{\rm in}$), respectively. The other crucial simulation parameters are provided in the captions of these figures. The ELM errors decrease significantly (approximately exponentially initially) with increasing number of boundary collocation points. On the other hand, increasing the number of interior collocation points in general only slightly improves the error in dimensions $d = 5$ and $d = 9$, and the error reduction is more significant in the lower dimension $d = 3$. These behaviors are similar to what has been observed with the Poisson equation in the previous subsection.

Finally, Figure 9 shows distributions of the ELM solution, the exact solution, and the ELM point-wise absolute error on three cross sections of the domain for $d = 5$: the x_1-x_3, x_2-x_4, and x_4-x_5 planes. For each plane, the other coordinates of the plane are set to zero. The ELM simulation parameters are listed in the figure caption, and the distributions are plotted on a set of 800×800 uniform grid points in these planes. It is evident that the ELM method has captured the solution very accurately.

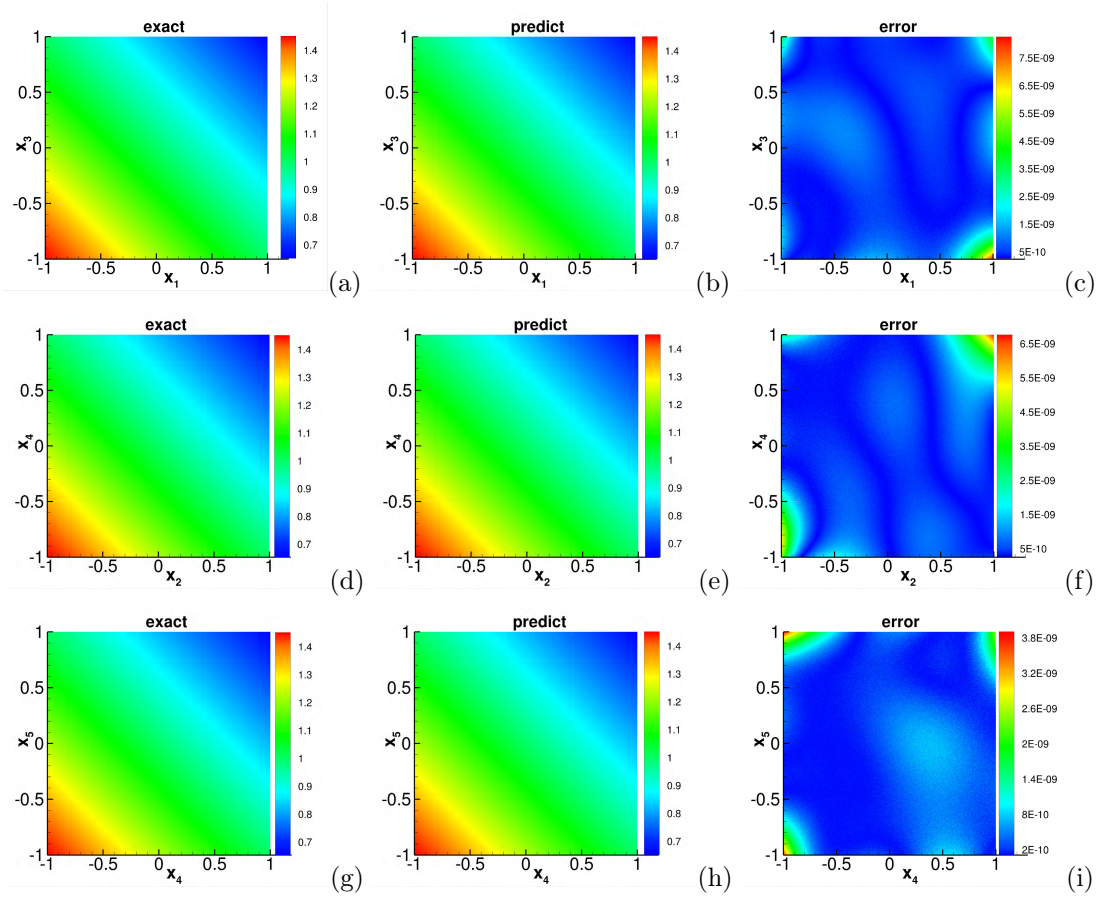


Figure 9: Nonlinear Poisson equation ($d = 5$): Distributions of the exact solution (left column), the ELM solution (middle column), and the ELM point-wise absolute error (right column) in the x_1-x_3 plane (top row), x_2-x_4 plane (middle row), and the x_4-x_5 plane (bottom row). NN architecture: $[d, 2500, 1]$; $(N_{\rm bc}, N_{\rm in}) = (100, 100)$.

3.1.3 Advection Diffusion Equation

We next test the ELM method using the high-dimensional advection-diffusion equation. Let $\Omega = [-1, 1]^d$ and $T = 1$. Consider the initial boundary value problem on the spatial-temporal domain $(x, t) \in \Omega \times [0, T]$,

$$\partial_t u - \nabla^2 u + R \cdot \nabla u = f(x, t), \quad (x, t) \in \Omega \times [0, T], \tag{44a}$$

$$u(x, t) = g(x, t), \quad (x, t) \in \partial\Omega \times [0, T], \tag{44b}$$

$$u(x, 0) = h(x), \quad x \in \Omega, \tag{44c}$$

where $R = \frac{1}{d}(1, 1, \ldots, 1)^T$, $f(x, t) = \frac{1}{d} \cos\left(\frac{1}{d}\sum_{i=1}^d x_i\right) \exp(-\frac{t}{d})$ on $\partial\Omega \times [0, T]$, and $h(x) = \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right)$. We employ the following analytic solution for this problem, $u(x, t) = \sin(\frac{1}{d}\sum_{i=1}^d x_i) \exp(-\frac{t}{d})$. $g(x, t)$ in (44b) is set according to this expression.

To simulate this problem with ELM, we treat the time variable t in the same way as the spatial coordinate x. We employ a neural network with an architecture, $\mathbf{M}_{\rm arch} = [d+1, M, 1]$, in which the $(d+1)$ input nodes represent (x, t) and the single output node represents the solution $u(x, t)$. So the problem has been effectively treated as a $(d+1)$-dimensional problem in the simulations. We enforce the initial condition on $N_{t_0} = 1000$ random collocation points on Ω at $t = 0$, and enforce the boundary condition on $N_{\rm bc}$ random collocation points on each of the boundaries $\partial\Omega \times [0, T]$. $N_{\rm in}$ denotes the number of random collocation points on the interior of $\Omega \times [0, T]$. After the neural network is trained, the e_{l^∞} and e_{l^2} errors are computed on a set of random test points from $\Omega \times \{T\}$ characterized by $(N_{\rm bc}^{(v)}, N_{\rm in}^{(v)}) = (100, 7000)$. Here $N_{\rm in}^{(v)}$ and $N_{\rm bc}^{(v)}$ denote

R_m	5E-3	1E-2	5E-2	1E-1	5E-1	1
e_{l^∞}	9.79E-4	9.33E-5	6.21E-8	2.15E-8	1.92E-5	4.82E-4
e_{l^2}	2.28E-4	8.17E-6	7.01E-9	1.38E-9	1.98E-6	8.54E-5

Table 5: Advection diffusion equation: Determining R_{m0} based on the procedure from Remark 6 for $d = 3$. NN architecture: $[d + 1, 1000, 1]$; $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (100, 100, 1000)$.

d	3	6	10
R_{m0}	0.1	0.05	0.05

Table 6: Advection diffusion equation: R_{m0} determined by the procedure from Remark 6 for several problem dimensions. The simulation parameters are the same as those in Table 5.

the number of random test points from the interior of $\Omega \times \{T\}$ and from each of the boundaries of $\Omega \times \{T\}$, respectively.

Table 5 shows the determination of R_{m0} using the procedure from Remark 6 for dimension $d = 3$, leading to $R_{m0} \approx 0.1$. Table 6 lists the R_{m0} values corresponding to several problem dimensions for the advection diffusion equation. In these tests for determining R_{m0}, we have employed an NN architecture $[d+1, 1000, 1]$, and the number of collocation points is characterized by $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (100, 100, 1000)$. In subsequent tests of this section, the hidden-layer coefficients are set to uniform random values from $[-R_{m0}, R_{m0}]$.

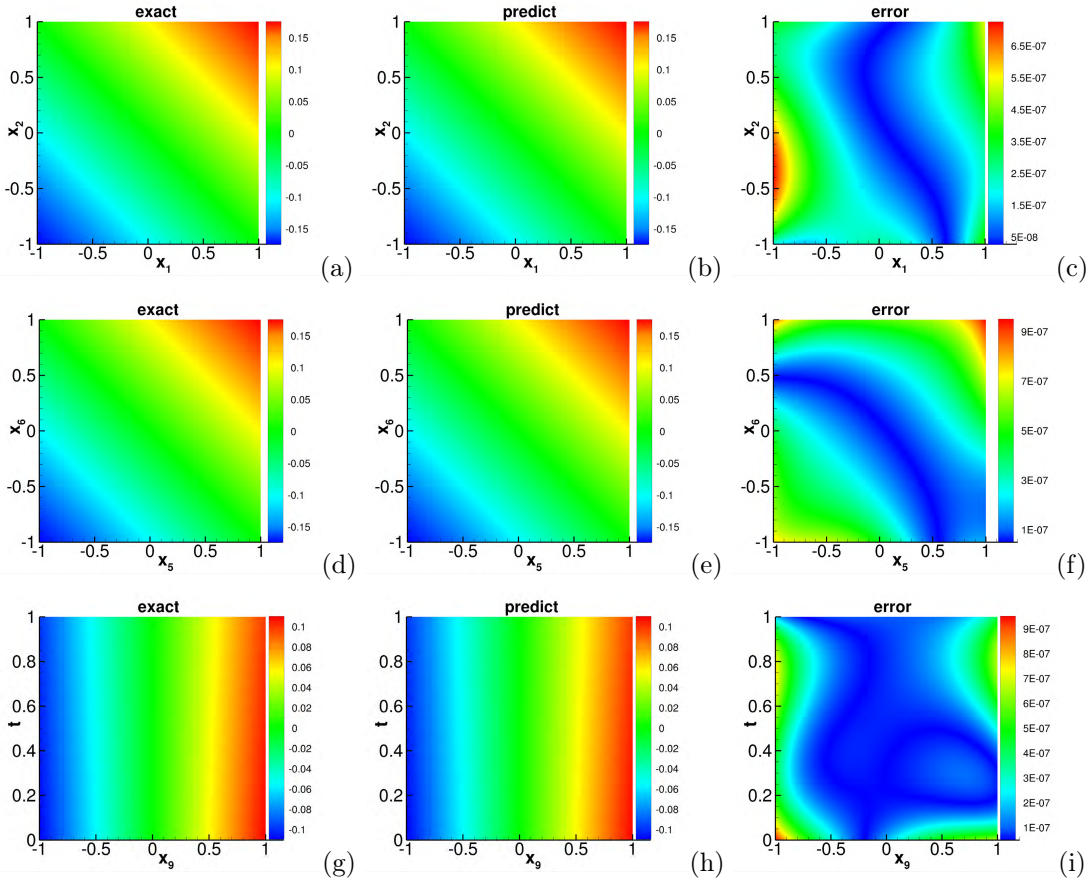
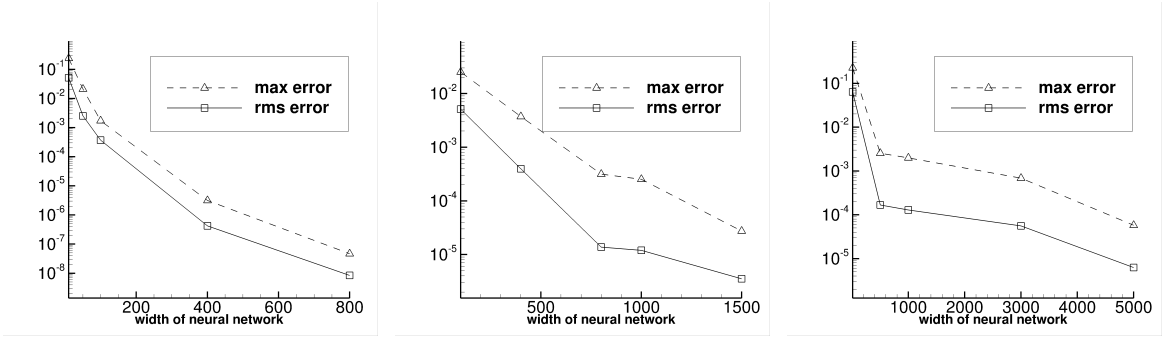
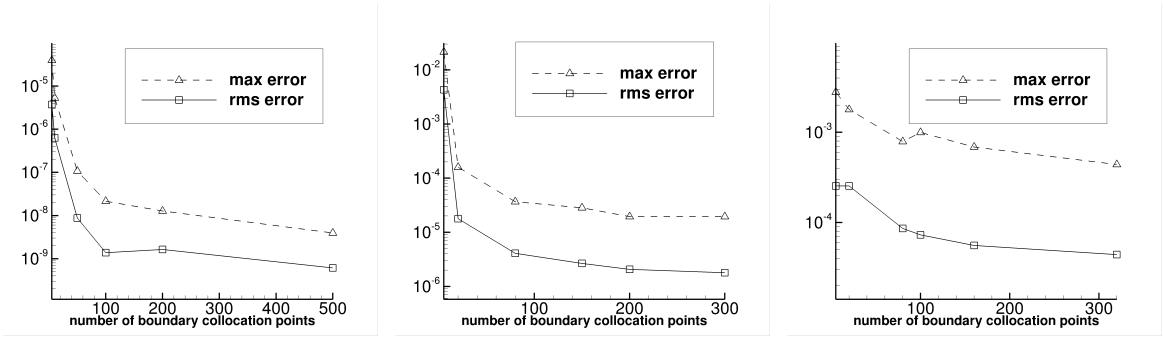


Figure 10: Advection diffusion equation ($d = 10$): Distributions of the exact solution (left column), the ELM solution (middle column), and the point-wise absolute error of ELM (right column) in selected cross sections of the domain, the x_1-x_2 plane (first row), the x_5-x_6 plane (middle row), and the x_9-t plane (bottom row). NN architecture: $[d + 1, 5000, 1]$; $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (160, 10, 1000)$. For each cross section, the other coordinates of the plane are in the middle of the domain in each direction.



(a) d=3 (b) d=6 (c) d=10

Figure 11: Advection diffusion equation: e_{l^∞} and e_{l^2} errors versus the number of training parameters (M) for problem dimensions (a) $d = 3$, (b) $d = 6$, and (c) $d = 10$. NN architecture: $[d + 1, M, 1]$; $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (200, 10, 1000)$ in (a), $(80, 10, 1000)$ in (b), and $(160, 10, 1000)$ in (c). M is varied in (a,b,c).



(a) d=3 (b) d=6 (c) d=10

Figure 12: Advection diffusion equation: e_{l^∞} and e_{l^2} errors versus the number of boundary collocation points ($N_{\rm bc}$) for dimensions (a) $d = 3$, (b) $d = 6$, and (c) $d = 10$. NN architecture: $[d + 1, M, 1]$ with $M = 1000$ in (a), $M = 2000$ in (b), and $M = 3000$ in (c). $(N_{\rm in}, N_{t_0}) = (10, 1000)$ in (a,b,c). $N_{\rm bc}$ is varied in (a,b,c).

Figure 10 shows distributions of the exact solution and the ELM solution, as well as the point-wise absolute error of the ELM solution, to the advection diffusion equation in dimension $d = 10$ on several cross sections of the spatial-temporal domain. These cross sections are the x_1-x_2 plane, the x_5-x_6 plane, and the x_9-t plane. Each plane is located in the middle of the spatial-temporal domain with respect to the rest of the coordinates. For example, the x_i-x_j plane is characterized by $t = T/2$ and $x_k = 0$ $(k \neq i, j)$ for $1 \leqslant k \leqslant d$, and the x_i-t plane is characterized by $x_k = 0$ $(k \neq i)$ for $1 \leqslant k \leqslant d$. The network architecture and the other simulation parameters are provided in figure caption. It is evident that ELM has captured the solution quite accurately, with the maximum error on the order of 10^{-7} on these cross sections.

Figure 11 illustrates the effect of the trainable parameters on the ELM accuracy. Here we show the e_{l^∞} and e_{l^2} errors of ELM versus the number of training parameters M in the neural network for solving the advection-diffusion equation in dimensions $d = 3$, 6 and 10. The network architecture is given by $[d+1, M, 1]$, where M is varied in the tests. The other simulation parameters are listed in the figure caption. The ELM errors can be observed to decrease dramatically (close to exponential rate) with increasing number of training parameters. The e_{l^2} (rms) error levels are on the order of 10^{-9} (for $d = 3$), and 10^{-6} (for $d = 6$ and 10) for the range of parameters tested here.

The effect of the boundary collocation points on the ELM accuracy is illustrated in Figure 12. The number of interior collocation points, on the other hand, has little (or much less) influence on the ELM results compared with the boundary points. Figure 12 shows the e_{l^∞} and e_{l^2} errors versus $N_{\rm bc}$ (collocation points on each boundary) for dimensions $d = 3$, 6 and 10. As $N_{\rm bc}$ increases, the errors appear to decrease

R_m	1E-3	5E-3	1E-2	5E-2	1E-1
e_{l^∞}	3.15E-4	1.48E-5	5.58E-6	2.51E-7	1.05E-6
e_{l^2}	2.31E-5	1.90E-6	2.97E-7	3.55E-8	1.71E-7

Table 7: KdV equation: Determining R_{m0} based on the procedure of Remark 6 for $d = 5$. NN architecture: $[d + 1, 2000, 1]$; $(N_{\text{in}}, N_{\text{bc}}, N_{t_0}) = (100, 100, 1000)$.

d	3	5	10
R_{m0}	0.05	0.05	0.05

Table 8: KdV equation: R_{m0} determined by the procedure from Remark 6 for several problem dimensions. In these tests the NN architecture and the number of collocation points are the same as those in Table 7 for $d = 3$ and 5. For $d = 10$, the NN architecture is $[d + 1, 3000, 1]$, and $(N_{\text{in}}, N_{\text{bc}}, N_{t_0}) = (10, 100, 1000)$.

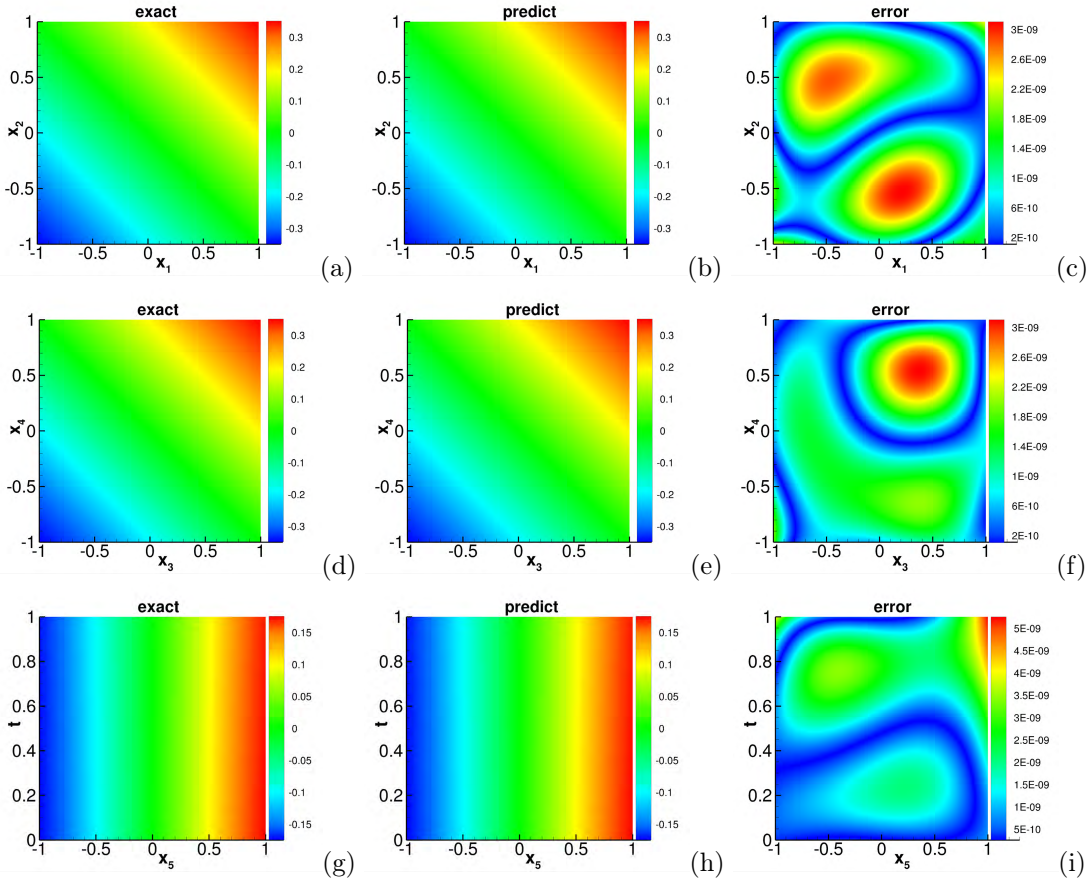
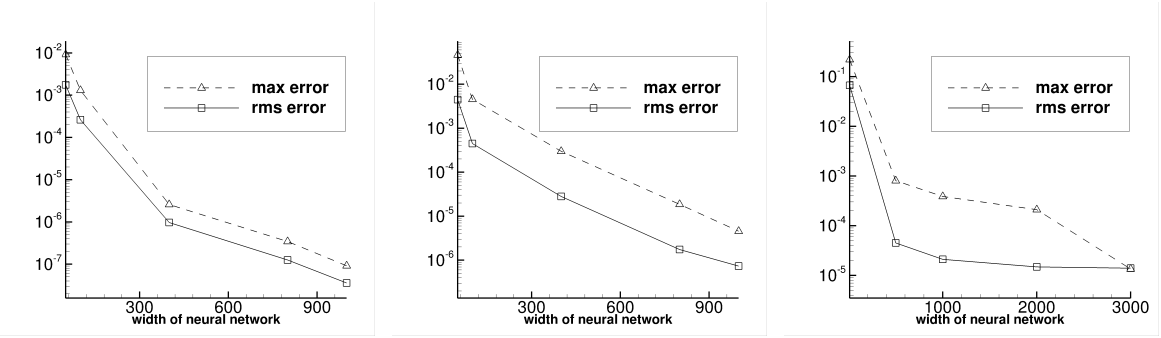


Figure 13: KdV equation ($d = 5$): Distributions of exact solution (left column), the ELM solution (middle column), and the point-wise absolute error of ELM (right column) on cross sections of the spatial-temporal domain: the x_1-x_2 plane (top row), x_3-x_4 plane (middle row), and x_5-t plane (bottom row). NN architecture: $[d + 1, 1000, 1]$; $(N_{\text{in}}, N_{\text{bc}}, N_{t_0}) = (10, 150, 1000)$.

approximately exponentially, and then level off when N_{bc} reaches a certain level.



(a) d=3 (b) d=5 (c) d=10

Figure 14: KdV equation: e_{l^∞} and e_{l^2} errors versus the number of training parameters (M) for dimensions (a) $d = 3$, (b) $d = 5$, and (c) $d = 10$. NN architecture: $[d + 1, M, 1]$; $(N_{\text{in}}, N_{t_0}) = (10, 1000)$ in (a,b,c); $N_{\text{bc}} = 200$ in (a), 150 in (b), and 100 in (c); M is varied in (a,b,c).

3.1.4 Korteweg-De Vries Equation

In this subsection we consider the Korteweg-De Vries (KdV) equation,

$$\partial_t u + \sum_{i=1}^{d} \partial^3_{x_i x_i x_i} u = f(x,t), \quad (x,t) \in \Omega \times [0,T], \tag{45a}$$

$$u(x,t) = g(x,t), \quad (x,t) \in \partial\Omega \times [0,T], \tag{45b}$$

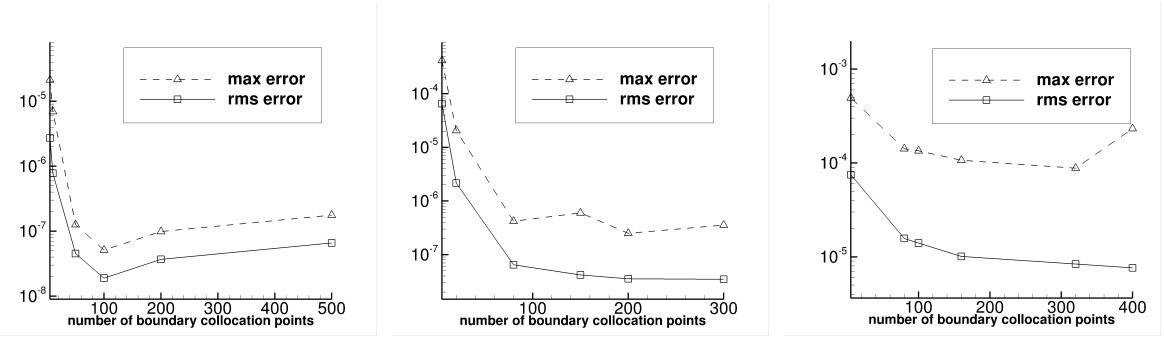
$$u(x,0) = h(x), \quad x \in \Omega, \tag{45c}$$

where $\Omega = [-1,1]^d$ and $T = 1$. In these equations $f(x,t) = -\frac{1}{d^2}\left[\sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right) + \cos\left(\frac{1}{d}\sum_{i=1}^d x_i\right)\right]\exp\left(-\frac{t}{d^2}\right)$, $g(x,t) = \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right)\exp\left(-\frac{t}{d^2}\right)$ on $\partial\Omega \times [0,T]$, and $h(x) = \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right)$ in Ω. This problem has an exact solution $u(x,t) = \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right)\exp\left(-\frac{t}{d^2}\right)$. The notations below follow those of the previous subsections.

Tables 7 documents the tests for determining the R_{m0} for $d = 5$ using the procedure from Remark 6, and Table 8 lists the resultant R_{m0} values corresponding to the dimensions $d = 3$, 5 and 10 from this procedure. The simulation parameters employed in these tests are provided in the captions of these figures. We use $R_m = R_{m0}$ for generating the random hidden-layer coefficients in the following tests with ELM.

Figure 13 provides an overview of distributions of the exact solution, the ELM solution, and the ELM point-wise absolute error in several 2D cross sections (x_1-x_2 plane, x_3-x_4 plane, x_5-t plane) of the spatial-temporal domain for the KdV equation in dimension $d = 5$. These cross sections are located in the middle of the domain with regard to the rest of the coordinates. The main simulation parameters for these results are listed in the figure caption. The ELM method has evidently captured the solution accurately, with an absolute error on the level of 10^{-9} in these cross sections.

The convergence behavior of the ELM method has been investigated and the test results are documented in Figures 14 and 15. These figures depict the e_{l^∞} and e_{l^2} errors for three problem dimensions ($d = 3$, 5 and 10) with respect to number of training parameters and the number of boundary collocation points (per boundary), respectively. The crucial simulation parameters in the tests are listed in the captions of these figures. With increasing number of training parameters in the network, the ELM errors decrease approximately exponentially. With respect to the number of boundary collocation points (N_{bc}), the ELM errors initially decreases approximately exponentially and gradually stagnates as N_{bc} reaches a certain level for $d = 3$ and 5. But for $d = 10$, the reduction in the ELM errors is not as significant as for the lower dimensions with increasing N_{bc}.



(a) d=3 (b) d=5 (c) d=10

Figure 15: KdV equation: e_{l^∞} and e_{l^2} errors versus the number of collocation points on each boundary ($N_{\rm bc}$) for several problem dimensions. NN architecture: $[d+1, M, 1]$; $M = 1000$ in (a), 2000 in (b), and 3000 in (c). $(N_{\rm in}, N_{t_0}) = (100, 1000)$ in (a), $(10, 1000)$ in (b,c). $N_{\rm bc}$ is varied in (a,b,c).

R_m	1E-3	5E-3	0.01	0.05	0.1
e_{l^∞}	2.65E-4	6.06E-5	2.94E-5	4.40E-5	6.64E-4
e_{l^2}	3.19E-5	5.19E-6	2.51E-6	4.11E-6	6.06E-5

Table 9: Poisson equation ($d = 7$): determining R_{m0} for the ELM/A-TFC method using the procedure from Remark 6. NN architecture: $[d, 3000, 1]$; $(N_{\rm bc}, N_{\rm in}) = (100, 1000)$.

3.2 Numerical Tests with the ELM/A-TFC Method

In this subsection we test the performance of the combined ELM/A-TFC method from Section 2.4 using several high-dimensional linear/nonlinear PDEs.

3.2.1 Poisson Equation

We employ the same Poisson problem as in Section 3.1.1 to test the ELM/A-TFC method. the governing equations are given by equations (42). The problem settings here follow those of Section 3.1.1. The notations below follow those of the test problems in Section 3.1.

Let us first determine the R_{m0} using the procedure from Remark 6 for generating the random hidden-layer coefficients with the ELM/A-TFC method. Table 9 shows the test using this procedure for dimension $d = 7$, leading to $R_{m0} \approx 0.01$. The R_{m0} values for different dimensions are listed in Table 10, which we will use for generating the random hidden-layer coefficients with ELM/A-TFC in subsequent tests.

An illustration of the distributions of the exact solution, the ELM/A-TFC solution, and the point-wise absolute error of the ELM/A-TFC solution for $d = 7$ is provided in Figure 16 for several 2D cross-sections of the domain (the x_1-x_2, x_4-x_5, x_6-x_7 planes). These planes are located in the middle of the domain with respect to the rest of the coordinates. These results are obtained using a network architecture $\mathbf{M}_{\rm arch} = [d, 3000, 1]$, with the collocation points characterized by $(N_{\rm bc}, N_{\rm in}) = (200, 10)$. The ELM/A-TFC results are observed to be quite accurate, with the maximum errors on the order of 10^{-6} or 10^{-7} in these cross sections.

The convergence behavior of ELM/A-TFC with respect to the number of training parameters is illustrated in Figure 17 for dimensions $d = 3$ and $d = 7$. Here the width of the single hidden layer in the network is

d	3	7
R_{m0}	0.1	0.01

Table 10: Poisson equation: R_{m0} for ELM/A-TFC determined by the procedure from Remark 6. The simulation parameters (NN architecture, collocation points) here follow those of Table 9, except that for $d = 3$ the NN architecture is $[d, 1000, 1]$.

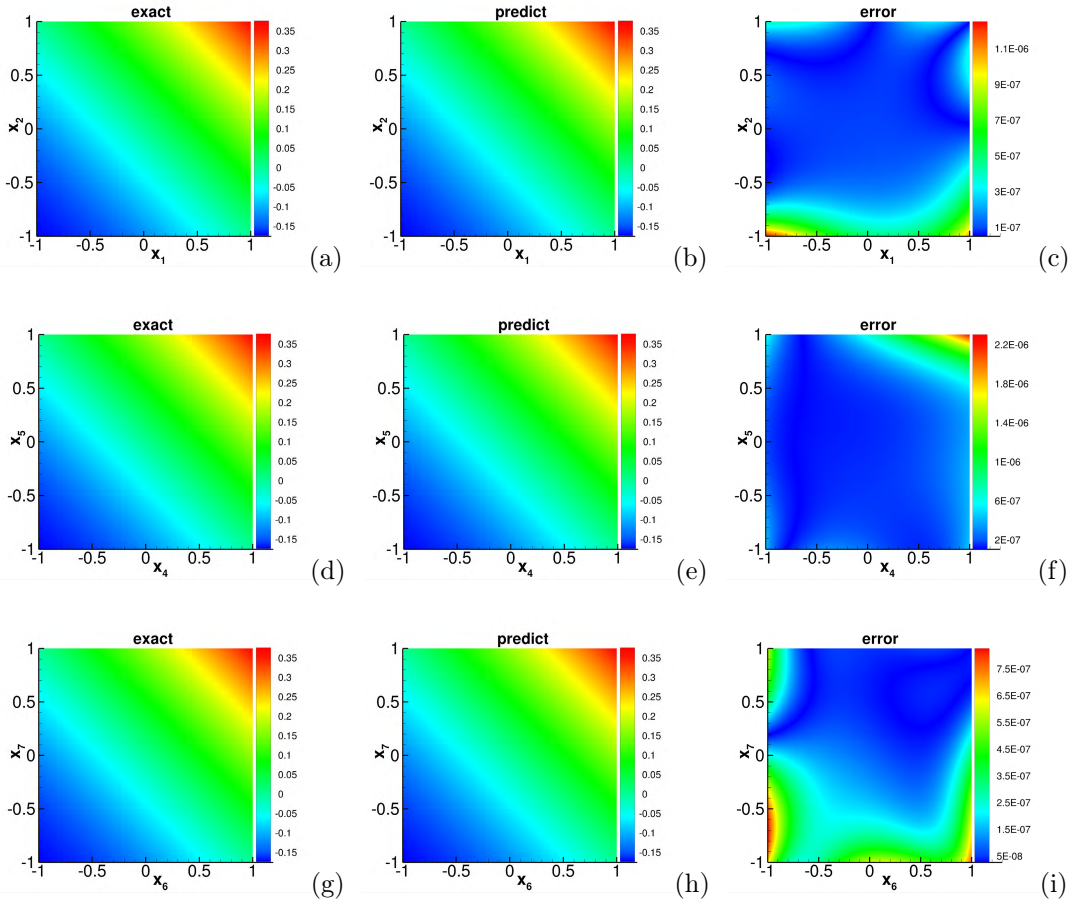
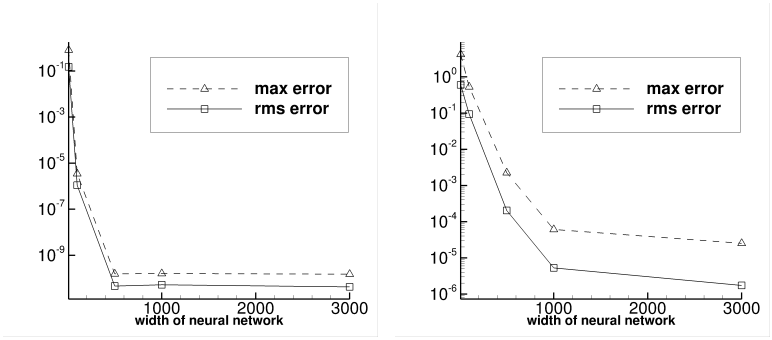


Figure 16: Poisson equation ($d = 7$): Distributions of the exact solution (left column), the ELM/A-TFC solution (middle column), and the point-wise absolute error of ELM/A-TFC (right column) in several 2D cross-sections of the domain (top row: x_1-x_2 plane; middle row: x_4-x_5 plane; bottom row: x_6-x_7 plane). NN architecture: $[d, 3000, 1]$; $(N_{\rm bc}, N_{\rm in}) = (200, 10)$.

varied, while the numbers of boundary/interior collocation points are fixed and listed in the figure caption. We observe an initial exponential decrease in the e_{l^∞} and e_{l^2} errors with increasing number of training parameters. Then the errors stagnate as the number of training parameters reaches a certain level.

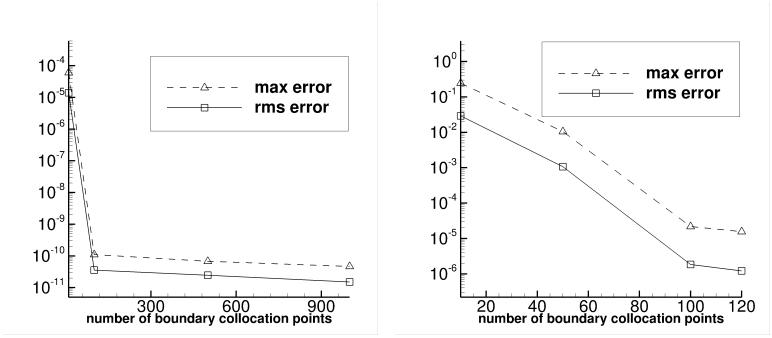
Figures 18 and 19 demonstrate the convergence behavior of ELM/A-TFC with respect to the number of boundary and interior collocation points, respectively. Here the neural network architecture is $\mathbf{M}_{\rm arch} = [d, M, 1]$, with a fixed $M = 1000$ for $d = 3$ and $M = 3000$ for $d = 7$. We observe an exponential decrease in the e_{l^∞} and e_{l^2} errors (before saturation), as the number of boundary collocation points ($N_{\rm bc}$) increases. On the other hand, the number of interior collocation points ($N_{\rm in}$) appears to have little effect on the ELM/A-TFC accuracy. These behaviors are similar to what have been observed with the ELM method in Section 3.1.

Finally we show a comparison between the ELM method and the ELM/A-TFC method for solving the Poisson equation. Table 11 lists the e_{l^∞} and e_{l^2} errors obtained by ELM and ELM/A-TFC corresponding to a set of $N_{\rm bc}$ values for two problem dimensions ($d = 3$ and 7). In these tests, the NN architecture is $\mathbf{M}_{\rm arch} = [d, M, 1]$ with $M = 1000$ for $d = 3$ and $M = 3000$ for $d = 7$, and the interior collocation points is fixed at $N_{\rm in} = 1000$. It is observed that the accuracy with ELM and ELM/A-TFC is generally comparable, and the ELM/A-TFC method appears to be slightly more accurate for lower dimensions. This can be attributed to the fact that the A-TFC resembles the full TFC more closely in lower dimensions. Therefore A-TFC enforces the boundary conditions more accurately (closer to TFC) in lower dimensions. On the other hand, we note that the computational effort and cost involved in ELM/A-TFC is generally higher than that



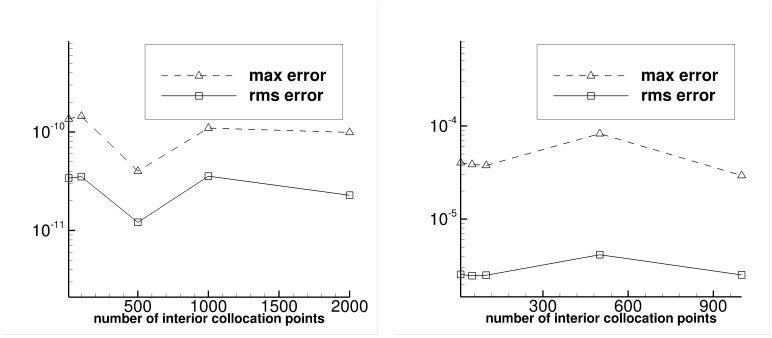
(a) d=3 (b) d=7

Figure 17: Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of training parameters (M) for dimensions (a) $d = 3$ and (b) $d = 7$. NN architecture: $[d, M, 1]$ (M varied); $(N_{\rm bc}, N_{\rm in}) = (100, 1000)$ in (a,b).



(a) dim=3 (b) dim=7

Figure 18: Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of boundary collocation points ($N_{\rm bc}$). NN architecture: $[d, M, 1]$, with $M = 1000$ in (a) and 3000 in (b). $N_{\rm in} = 1000$ in (a,b), while $N_{\rm bc}$ is varied.



(a) dim=3 (b) dim=7

Figure 19: Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of interior collocation points ($N_{\rm in}$). NN architecture: $[d, M, 1]$, with $M = 1000$ in (a) and 3000 in (b). $N_{\rm bc} = 100$ in (a,b), while $N_{\rm in}$ is varied.

Dimension	$N_{\rm bc}$	ELM		ELM/A-TFC	
		e_{l^∞}	e_{l^2}	e_{l^∞}	e_{l^2}
$d=3$	10	1.16E-4	1.11E-5	6.08E-5	1.40E-5
	100	1.03E-9	7.62E-11	1.10E-10	3.57E-11
	500	2.96E-10	3.83E-11	6.76E-11	2.47E-11
	1000	2.62E-10	3.58E-11	4.69E-11	1.50E-11
$d=7$	80	1.12E-5	1.09E-6	9.81E-5	7.24E-6
	90	9.26E-6	7.30E-7	6.85E-5	4.64E-6
	100	8.16E-6	6.64E-7	2.96E-5	2.51E-6
	110	7.30E-6	6.04E-7	2.48E-5	2.21E-6

Table 11: Poisson equation: Comparison of e_{l^∞} and e_{l^2} errors obtained with ELM and ELM/A-TFC corresponding to a set of boundary collocation points ($N_{\rm bc}$). NN architecture: $[d, M, 1]$, with $M = 1000$ for $d = 3$ and with $M = 3000$ for $d = 7$. $N_{\rm in} = 1000$ in all cases.

R_m	0.01	0.05	0.1	0.5	1.0
e_{l^∞}	5.82E-6	5.21E-9	2.92E-9	3.47E-6	4.85E-1
e_{l^2}	1.64E-6	9.78E-10	5.72E-10	7.34E-7	6.71E-2

Table 12: Nonlinear Poisson equation ($d = 3$): Determining R_{m0} for the ELM/A-TFC method based on the procedure from Remark 6. NN architecture: $[d, 500, 1]$; $(N_{\rm bc}, N_{\rm in}) = (100, 10)$.

of ELM, because of the computations associated with the A-TFC terms.

3.2.2 Nonlinear Poisson Equation

In this subsection we test the ELM/A-TFC method using the nonlinear Poisson problem from Section 3.1.2, under the same problem settings and parameters.

Tables 12 and 13 show the tests for determining the R_{m0} using the procedure from Remark 6. The results lead to $R_{m0} \approx 0.1$ for dimension $d = 3$ and $R_{m0} \approx 0.01$ for $d = 7$. These values are employed for generating the hidden-layer coefficients in the neural network in the subsequent simulations.

Figure 20 shows distributions of the exact solution, the ELM/A-TFC solution, and the point-wise absolute error of ELM/A-TFC in several cross sections of the domain for the nonlinear Poisson equation in dimension $d = 7$. The ELM/A-TFC results are obtained with an NN architecture $\mathbf{M}_{\rm arch} = [d, 1000, 1]$ and the random collocation points characterized by $(N_{\rm bc}, N_{\rm in}) = (50, 10)$. The ELM/A-TFC method has captured the solution accurately, with the maximum error on the order 10^{-5} in the x_1-x_3 and x_3-x_6 planes and on the order of 10^{-7} in the x_6-x_7 plane.

The convergence behavior of the ELM/A-TFC method is illustrated by Figures 21 and 22 for problem dimensions $d = 3$ and $d = 7$. Figure 21 shows the e_{l^∞} and e_{l^2} errors as a function of the number of training parameters in the neural network. In this set of tests, the number of boundary/interior collocation points is fixed while the number of training parameters is varied. We can observe a rapid decrease (approximately exponential) in the ELM/A-TFC errors as the number of training parameters increases (before saturation). Figure 22 shows the e_{l^∞} and e_{l^2} errors as a function of the number of boundary collocation points ($N_{\rm bc}$). In this set of tests the number of training parameters and the number of interior collocation points are fixed, while the number of boundary collocation points ($N_{\rm bc}$) is varied systematically. The ELM/A-TFC errors initially decrease rapidly as $N_{\rm bc}$ increases, and then level off when $N_{\rm bc}$ increases beyond a certain level. We would like to further point out that the number of interior collocation points has little effect on the ELM/A-TFC accuracy (result not shown here), similar to what has been observed with the ELM method.

d	3	7
R_{m0}	0.1	0.01

Table 13: Nonlinear Poisson equation: R_{m0} for ELM/A-TFC determined by the procedure from Remark 6. The simulation parameters here for $d = 3$ follow those of Table 10. For $d = 7$, NN architecture: $[d, 3000, 1]$, $(N_{\mathbf{c}}, N_{\rm in}) = (300, 100)$.

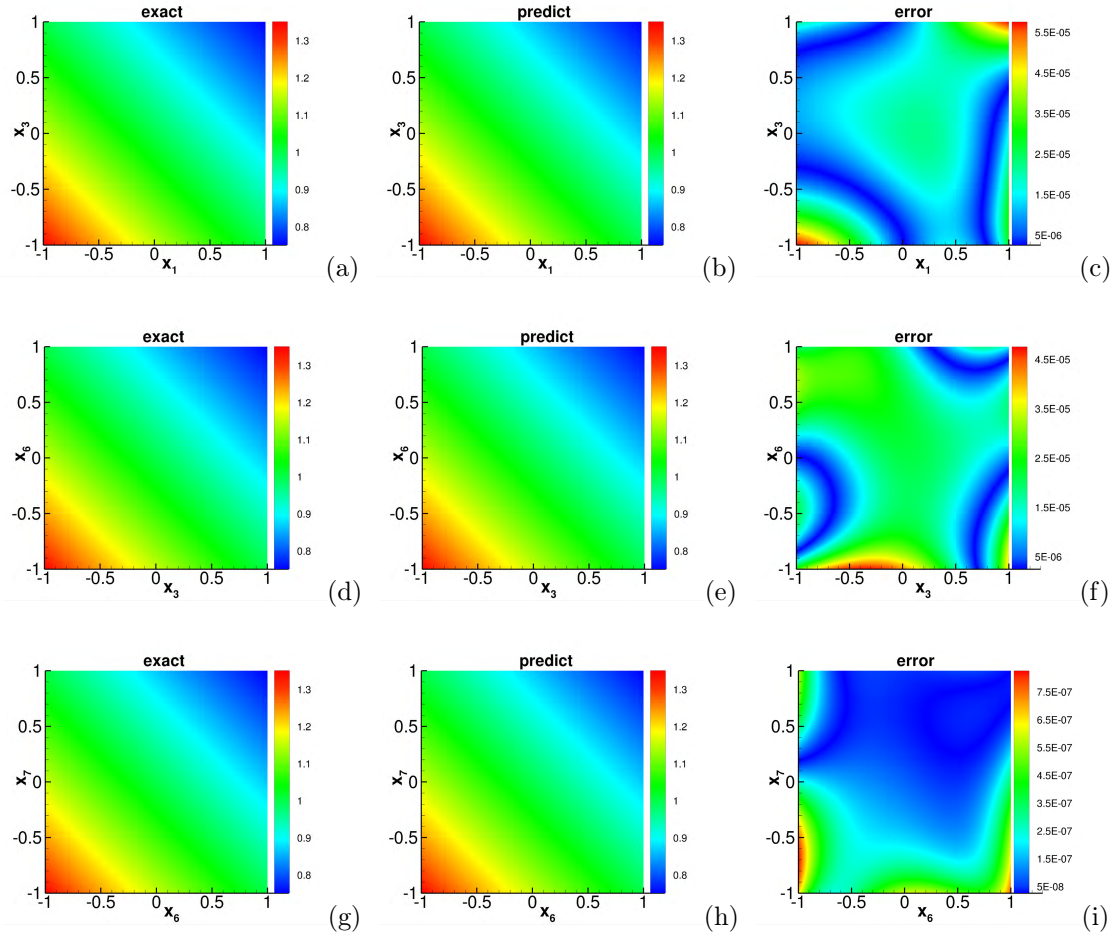
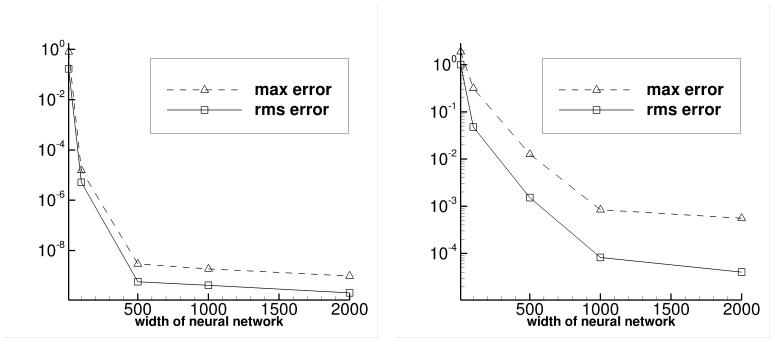
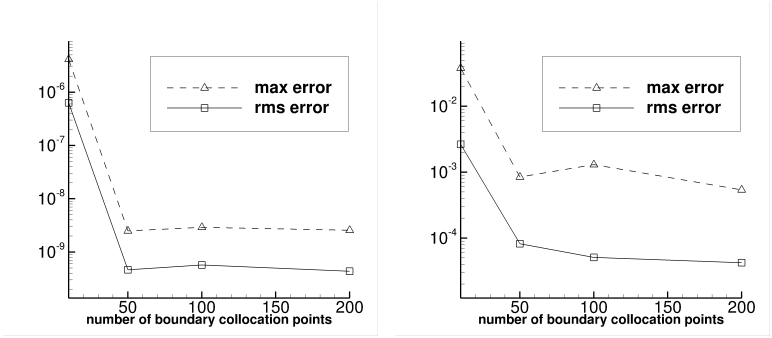


Figure 20: Nonlinear Poisson equation ($d = 7$): Distributions of the exact solution (left column), the ELM/A-TFC solution (middle column), and the point-wise absolute error of ELM/A-TFC in several cross sections (top row: x_1-x_3 plane; middle row: x_3-x_6 plane; bottom row: x_6-x_7 plane) of the domain. These cross sections are located in the middle of the domain with respect to the rest of the coordinates. NN architecture: $[d, 1000, 1]$, $(N_{\rm bc}, N_{\rm in}) = (50, 10)$.



(a) d=3 (b) dim=7

Figure 21: Nonlinear Poisson equation: e_{l^∞} and e_{l^2} errors versus the number of training parameters (M) in the ELM/A-TFC network. NN architecture: $[d, M, 1]$ (with M varied); $(N_{\rm bc}, N_{\rm in}) = (100, 10)$ in (a) and $(50, 100)$ in (b).



(a) d=3 (b) d=7

Figure 22: Nonlinear Poisson equation: e_{l^∞} and e_{l^2} errors of ELM/A-TFC versus the number of boundary collocation points ($N_{\rm bc}$). NN architecture: $[d, M, 1]$, with $M = 500$ in (a) and $M = 1000$ in (b); $N_{\rm in} = 10$ in (a) and $N_{\rm in} = 100$ in (b); $N_{\rm bc}$ varied in (a,b).

R_m	1E-3	1E-2	5E-2	1E-1	5E-1	1
e_{l^∞}	6.05E-3	4.15E-4	3.56E-6	5.32E-7	2.64E-5	8.21E-4
e_{l^2}	1.16E-3	6.18E-5	6.37E-7	9.35E-8	6.03E-6	1.54E-4

Table 14: Heat equation: Determination of R_{m0} for ELM/A-TFC using the procedure from Remark 6 for $d = 3$. NN architecture: $[d + 1, 1000, 1]$; $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (100, 100, 1000)$.

3.2.3 Heat Equation

We next consider the domain $\Omega = [-1, 1]^d$ and the heat equation on $\Omega \times [0, T]$ (with $T = 1$),

$$\partial_t u - \Delta u = f(x,t), \quad (x,t) \in \Omega \times [0,T], \tag{46}$$

$$u(x,t) = g(x,t), \quad (x,t) \in \partial\Omega \times [0,T], \tag{47}$$

$$u(x,0) = h(x), \quad x \in \Omega, \tag{48}$$

where $f(x,t) = (\frac{1}{d}-1)\cos\left(\frac{1}{d}\sum_{i=1}^d x_i\right)\exp(-t)$, $g(x,t) = \cos\left(\frac{1}{d}\sum_{i=1}^d x_i\right)\exp(-t)$, and $h(x) = \cos\left(\frac{1}{d}\sum_{i=1}^d x_i\right)$. This problem has the exact solution $u(x,t) = \cos\left(\frac{1}{d}\sum_{i=1}^d x_i\right)\exp(-t)$.

The simulation settings and the notations here follow those of Section 3.1.3. We employ a neural network architecture $\mathbf{M}_{\rm arch} = [d + 1, M, 1]$, where the $(d + 1)$ input nodes denote (x, t) and M is the number of training parameters in the network. $N_{\rm bc}$, $N_{\rm in}$ and N_{t_0} denote the number of random collocation points on each of the boundary of $\partial\Omega \times [0, T]$, on the interior of $\Omega \times [0, T]$, and on Ω at $t = 0$, respectively.

Table 14 summarizes the tests for determining the R_{m0} with the ELM/A-TFC method using the procedure from Remark 6 for $d = 3$, which lead to $R_{m0} \approx 0.1$. Table 15 lists the R_{m0} values for the problem dimensions we have considered for this problem. The hidden-layer coefficients are set to uniform random values generated on $[-R_{m0}, R_{m0}]$ in the subsequent simulations.

Figure 23 illustrates the distributions of the exact solution, the ELM/A-TFC solution, and the point-wise absolute error of ELM/A-TFC in several cross sections of the spatial-temporal domain for the problem dimension $d = 7$. It is observed that the ELM/A-TFC method has captured the solution fairly accurately, with the maximum error on the order of 10^{-4} in these cross sections.

Figures 24 and 25 demonstrate the convergence behavior of the ELM/A-TFC method with respect to the number of training parameters and the number of boundary collocation points. The simulation parameters

d	3	7
R_{m0}	0.1	0.005

Table 15: Heat equation: R_{m0} for ELM/A-TFC determined by the procedure from Remark 6. The simulation parameters here follow those of Table 14.

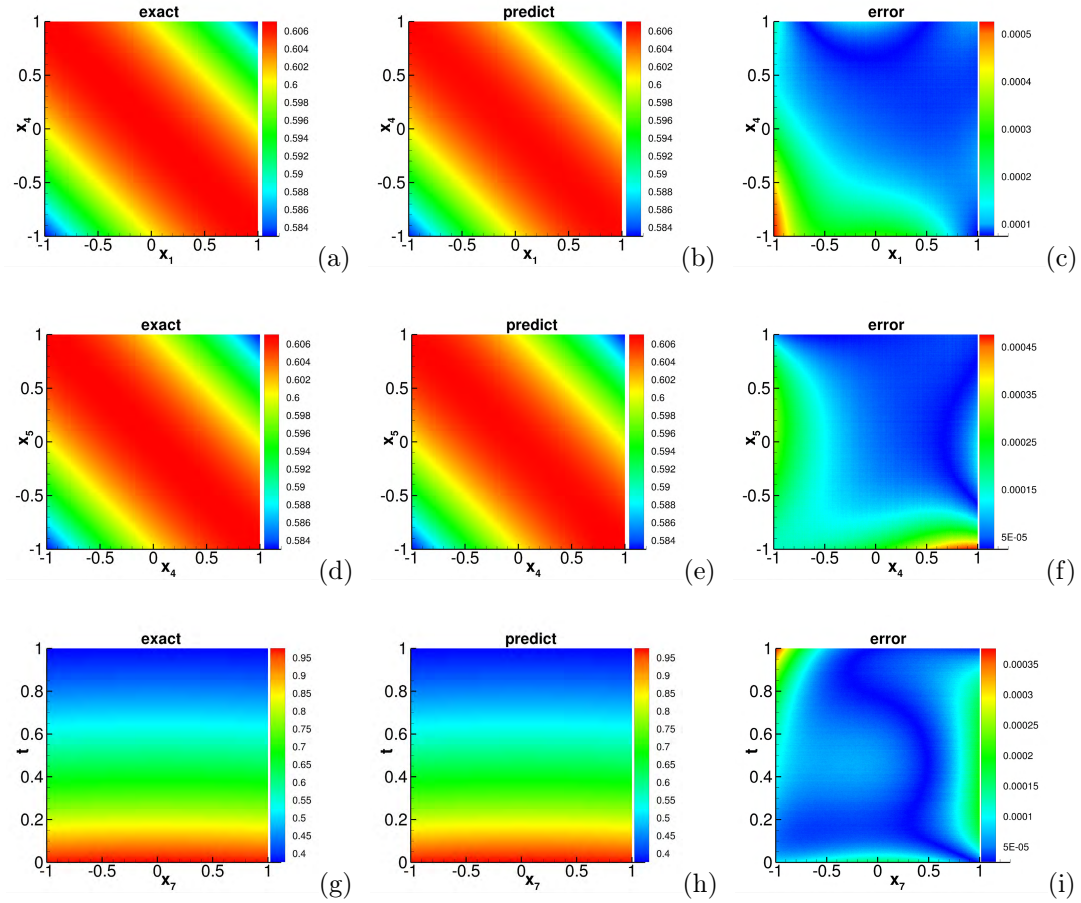
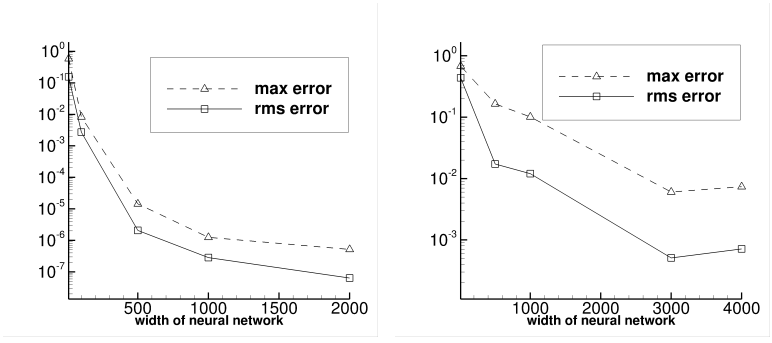
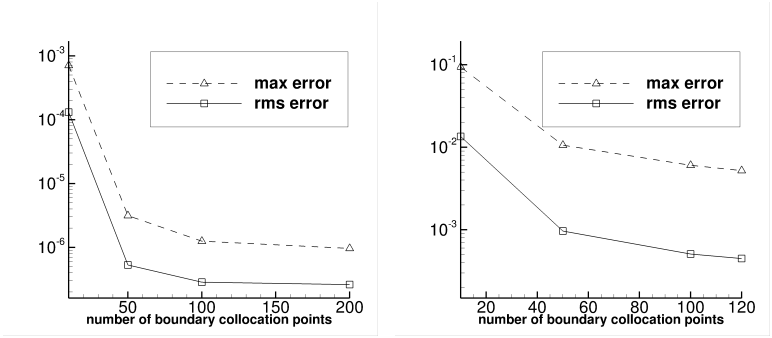


Figure 23: Heat equation: ($d = 7$): Distributions of the exact solution (left column), the ELM/A-TFC solution (middle column), and the point-wise absolute error of ELM/A-TFC in several cross sections (top row: x_1-x_4 plane; middle row: x_4-x_5 plane; bottom row: x_7-t plane) of the spatial-temporal domain $\Omega \times [0, T]$. These cross sections are located in the middle of the spatial-temporal domain with respect to the rest of coordinates. NN architecture: $[d + 1, 3000, 1]$; $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (120, 10, 1000)$.



(a) d=3 (b) d=7

Figure 24: Heat equation: e_{l^∞} and e_{l^2} errors of ELM/A-TFC versus the number of training parameters (M). NN architecture: $[d + 1, M, 1]$ (with M varied); $(N_{\rm bc}, N_{\rm in}, N_{t_0}) = (100, 100, 1000)$ in (a) and $(100, 10, 1000)$ in (b).



(a) dim=3 (b) dim=7

Figure 25: Heat equation: e_{l^∞} and e_{l^2} errors of ELM/A-TFC versus the number of boundary collocation points ($N_{\rm bc}$). NN architecture: $[d+1, M, 1]$, with $M = 1000$ in (a) and $M = 3000$ in (b). $(N_{\rm in}, N_{t_0}) = (100, 1000)$ in (a) and $(10, 1000)$ in (b). $N_{\rm bc}$ is varied in (a,b).

Dimension	Classical PINN			Current (ELM)		
	e_{l^∞}	e_{l^2}	train-time(sec)	e_{l^∞}	e_{l^2}	train-time(sec)
$d = 3$	3.84E-3	6.90E-4	326.8	2.34E-10	4.42E-12	2.6
$d = 5$	1.25E-2	1.45E-3	552.5	1.30E-8	5.20E-10	3.5
$d = 7$	8.12E-2	2.85E-3	688.1	8.16E-6	6.64E-7	11.2
$d = 9$	8.56E-1	1.16E-1	964.0	3.95E-5	5.26E-6	14.6

Table 16: Poisson equation: Comparison of e_{l^∞} and e_{l^2} errors and the network training time between ELM and classical PINN. See the text for simulation parameters.

in these tests have been provided in the figure captions. The characteristics are similar to what have been observed for other test problems in previous subsections. With regard to the number of interior collocation points, we again observe that it has little influence on the accuracy of ELM/A-TFC (result not shown here).

3.3 Comparison with PINN

We next compare the current ELM method with the classical physics-informed neural network (PINN) method [78] for the Poisson equation of Section 3.1.1 and the nonlinear Poisson equation of Section 3.1.2 for a range of problem dimensions. In PINN all the network parameters are trained, and the loss function consists of two terms, the term for the PDE residual and the one for the residual of the boundary conditions. We employ the penalty coefficients $(1-p_{\rm bc})$ and $p_{\rm bc}$ in front of the loss terms for the PDE and the boundary conditions, respectively, where $p_{\rm bc} \in (0, 1)$ is a constant. The Adam optimizer is used to train the neural network in PINN. Our PINN implementation is also based on the Tensorflow and Keras libraries.

With PINN, we have varied the random initialization of the weight/bias coefficients, the neural network architecture, the learning rate, the learning rate schedule, and the penalty coefficient $p_{\rm bc}$ systematically for training the neural network. The PINN results reported below are the best we have obtained in these tests. It should be noted that much poorer PINN results (not used for comparison or shown here) have been

Dimension	Classical PINN			Current (ELM)		
	e_{l^∞}	e_{l^2}	train-time(sec)	e_{l^∞}	e_{l^2}	train-time(sec)
$d = 3$	2.23E-3	4.77E-4	1977.4	8.10E-10	1.20E-11	29.8
$d = 5$	2.52E-3	3.75E-4	2526.3	3.20E-8	1.13E-9	112.5
$d = 7$	1.89E-2	1.24E-3	3517.9	3.96E-5	3.23E-6	220.9
$d = 9$	4.781E-2	1.33E-2	5036.0	1.61E-4	1.27E-5	678.8

Table 17: Nonlinear Poisson equation: Comparison of e_{l^∞} and e_{l^2} errors and the network training time between ELM and classical PINN. See the text for simulation parameters.

obtained in these tests.

Table 16 compares the e_{l^∞} and e_{l^2} errors, as well as the network training time (in seconds), obtained with PINN and with the current ELM method for solving the Poisson problem from Section 3.1.1 in dimensions ranging from $d = 3$ to $d = 9$. The PINN results are obtained with a network architecture $[d, 30, 30, 30, 30, 30, 30, 1]$ (tanh activation function), a penalty coefficient $p_{\rm bc} = 0.99$, and the random collocation points characterized by $(N_{\rm bc}, N_{\rm in}) = (100, 3000)$ for the boundary and interior of the domain for dimensions $d = 3$, 5 and 7 and $(N_{\rm bc}, N_{\rm in}) = (200, 5000)$ for dimension $d = 9$. A staircase learning rate schedule has been employed with PINN, starting with a learning rate 0.01 and decaying by a rate 0.5 every 500 epochs. The PINN has been trained for a total of 5000 epochs for the Poisson problem. The ELM results are obtained using a network architecture $[d, 2000, 1]$ for dimensions $d = 3$ and 5 and $[d, 3000, 1]$ for dimensions $d = 7$ and 9. The ELM random collocation points are characterized by $(N_{\rm bc}, N_{\rm in}) = (100, 200)$ for $d = 3$ and 5 and $(N_{\rm bc}, N_{\rm in}) = (100, 1000)$ for $d = 7$ and 9 for the boundary and interior of the domain. For generating the ELM random hidden-layer coefficients we employ $R_m = 0.5$ for $d = 3$, $R_m = 0.05$ for $d = 5$ and $d = 7$, and $R_m = 0.001$ for $d = 9$. Compared with PINN, the current ELM method produces significantly more accurate results with a much smaller training time. For example, for dimension $d = 5$ the PINN method produces an error on the order of $10^{-3} \sim 10^{-2}$ with a network training time close to 600 seconds. In contrast, for this case the ELM method produces an error on the order of $10^{-10} \sim 10^{-8}$ with a network training time around 3 or 4 seconds.

Table 17 compares the e_{l^∞} and e_{l^2} errors, as well as the network training time, obtained by PINN and ELM for the nonlinear Poisson problem from Section 3.1.2 for dimensions ranging from $d = 3$ to $d = 9$. The PINN results correspond to a network architecture $[d, 30, 30, 30, 30, 30, 30, 1]$ (tanh activation function), a penalty coefficient $p_{\rm bc} = 0.99$, a staircase learning rate schedule starting with a learning rate 0.01 and decaying at a rate 0.8 every 1000 epochs, a set of random collocation points characterized by $(N_{\mathbf{c}}, N_{\rm in}) = (100, 3000)$ on the boundary and interior of the domain, and a total of 20000 training epochs. The ELM results correspond to a network architecture $[d, 2000, 1]$, and a set of random collocation points characterized by $(N_{\rm bc}, N_{\rm in}) = (100, 200)$ for dimensions $d = 3$, 5 and 9 and $(N_{\rm bc}, N_{\rm in}) = (100, 100)$ for $d = 7$ on the boundary and interior of the domain. We employ $R_m = 0.5$ for $d = 3$, $R_m = 0.05$ for $d = 5$ and $d = 7$, and $R_m = 0.001$ for $d = 9$ for generating the ELM random hidden-layer coefficients. The results here signify the considerably higher accuracy and less network training cost of ELM, when compared with PINN, for the nonlinear problem. For example, for dimension $d = 5$ the PINN method achieves an error level on the order of $10^{-4} \sim 10^{-3}$ with a training time around 2500 seconds, while the ELM method achieves an error on the order of $10^{-9} \sim 10^{-8}$ with a network training time around 110 seconds.

4 Concluding Remarks

In this paper we have presented two methods for computing high-dimensional PDEs based on randomized neural networks. These methods are motivated by the theoretical result established in the literature that the ELM-type randomized NNs can effectively approximate high-dimensional functions, with a rate of convergence independent of the function dimension in the sense of expectations.

The first method extends the ELM approach, and its local variant locELM, developed in a previous work for low-dimensional problems to linear/nonlinear PDEs in high dimensions. We represent the solution field to the high-dimensional PDE problem by a randomized NN, with its hidden-layer coefficients assigned to random values and fixed and its output-layer coefficients trained. Enforcing the PDE problem on a set of collocation points randomly distributed on the interior/boundary of the domain leads to an algebraic system of equations, which is linear for linear PDE problems and nonlinear for nonlinear PDE problems, about the ELM trainable parameters. By seeking a least squares solution to this algebraic system, attained by either a linear or a nonlinear least squares method, we can determine the values for the training parameters and complete the network training. ELM can be combined with domain decomposition and local randomized NNs for solving high-dimensional PDEs, leading to a local variant of this method. In this case, domain decomposition is performed along a maximum of two designated directions for a d-dimensional problem, and the PDE problem, together with appropriate continuity conditions, is enforced on the random collocation points on each sub-domain and the shared sub-domain boundaries.

Compared with the ELM for low-dimensional problems, the difference of the method here for high-

dimensional PDEs lies in at least two aspects. First, the collocation points employed for training the ELM network for high-dimensional PDEs are randomly generated on the interior and the boundaries of the domain (or the sub-domains), and the number of interior collocation points has little (essentially no) effect on the ELM accuracy in high dimensions. In contrast, for low-dimensional PDE problems the ELM neural network is trained largely on grid-based collocation points (e.g. uniform grid points, or quadrature points), and the number of interior collocation points critically influences the ELM accuracy. Second, with the local variant of ELM (plus domain decomposition) for solving high-dimensional PDEs, the domain is only decomposed along a maximum of $\mathcal{M}$ directions, where $\mathcal{M}$ is a prescribed small integer ($\mathcal{M} = 2$ in this paper), so as for the method to be feasible in high dimensions. This is an issue not present for low-dimensional PDEs.

The second method (ELM/A-TFC) combines the ELM approach and an approximate variant of TFC (A-TFC) for solving high-dimensional PDEs. While TFC provides a systematic approach to enforce the boundary/initial conditions, the number of terms involved in TFC constrained expression grows exponentially as the problem dimension increases, rendering it infeasible for high-dimensional problems. By noting that the TFC constrained expression can be decomposed into a hierarchical form, we introduce the A-TFC by retaining only the dominant terms in the constrained expression. A-TFC avoids the exponential growth in the number of terms of TFC and is feasible for high-dimensional PDEs. On the other hand, the A-TFC constrained expression does not unconditionally satisfy the boundary/initial conditions for an arbitrary free function in the expression. However, the conditions that the free function in the A-TFC constrained expression needs to fulfill, in order to satisfy the boundary/initial conditions, involve functions of simpler forms, which in some sense can be considered as an effective linearization of those involved in the original boundary/initial conditions. A-TFC carries a level of benefit of TFC for enforcing the boundary/initial conditions and is simultaneously suitable for high-dimensional problems. With the ELM/A-TFC method, we reformulate the high-dimensional PDE problem using the A-TFC constrained expression, and attain a transformed problem about the free function involved in the A-TFC expression. We represent this free function by ELM, and determine the ELM trainable parameters by the linear or nonlinear least squares method in a fashion analogous to the first method. After the free function is determined by the ELM network, the solution field to the original high-dimensional PDE problem is then computed by the A-TFC constrained expression.

The two methods have been tested numerically using a number of linear/nonlinear stationary/dynamic PDEs for a range of problem dimensions. The method has also been compared with the PINN method. We have the following observations from these numerical results:

- Both the ELM method and the ELM/A-TFC method produce accurate solutions to high-dimensional PDEs, in particular with their errors reaching levels not far from the machine accuracy for relatively lower dimensions.
- Both methods exhibit a clear sense of convergence with respect to the number of trainable parameters and the number of boundary collocation points. Their errors decrease rapidly (exponentially or nearly exponentially) for an initial range of parameter values (before saturation).
- The number of interior collocation points appears to have a minimal (essentially no) effect on the accuracy of ELM and ELM/A-TFC for high-dimensional PDEs.
- For a given PDE, the problem becomes more challenging to compute with increasing dimension, in the sense that the errors of both methods in higher dimensions generally appear somewhat worse than in lower dimensions, at least with the range of parameter values tested in this work.
- The error levels obtained by the ELM method and the ELM/A-TFC method are generally comparable, with ELM/A-TFC appearing slightly better in lower dimensions. On the other hand, the ELM/A-TFC method generally involves a larger computational effort and cost than ELM, due to the A-TFC constrained expression.
- The current method exhibits a clear advantage compared with PINN for solving high-dimensional PDEs, and achieves a significantly better accuracy under markedly smaller training time than the latter.

The simulation results signify that the ELM-based methods developed herein are effective for computational PDEs in high dimensions.

Acknowledgment

This work was partially supported by the US National Science Foundation (DMS-2012415).

References

[1] Diab W Abueidda, Qiyue Lu, and Seid Koric. Meshless physics-informed deep learning method for three-dimensional solid mechanics. *International Journal for Numerical Methods in Engineering*, 122(23):7182–7201, 2021.

[2] P.A. Alaba, S.I. Popoola, L. Olatomiwa, M.B. Akanle, O.S. Ohunakin, E. Adetiba, O.D. Alex, A.A.A. Atayero, and W.M.A.W. Daud. Towards a more efficient and cost-sensitive extreme learning machine: a state-of-the-art review of recent trend. *Neurocomputing*, 350:70–90, 2019.

[3] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[4] Christian Beck, Weinan E, and Arnulf Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *J. Nonlinear Sci.*, 29(4):1563–1619, 2019.

[5] Richard E Bellman. *Dynamic programming*. Princeton university press, 2010.

[6] Julius Berner, Philipp Grohs, and Arnulf Jentzen. Analysis of the generalization error: empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *SIAM J. Math. Data Sci.*, 2(3):631–657, 2020.

[7] A. Bjorck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[8] H.A.T. Braake and G.V. Straten. Random activation weight neural net (RAWN) for fast non-iterative training. *Eng. Applic. Artif. Intell.*, 8:71–80, 1995.

[9] F. Calabro, S. Cuomo, D. di Serafino, G. Izzo, and E. Messina. Time discretization in the solution of parabolic pdes with anns. *Applied Mathematics and Computation*, 458:128230, 2023.

[10] F. Calabro, G. Fabiani, and C. Siettos. Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients. *Computer Methods in Applied Mechanics and Engineering*, 387:114188, 2021.

[11] W. Cao, X. Wang, Z. Ming, and J. Gao. A review on neural networks with random weights. *Neurocomputing*, 275:278–287, 2018.

[12] 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. *Proceedings of Machine Learning Research*, 107:512–536, 2020.

[13] Jérôme Darbon and Stanley Osher. Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere. *Research in the Mathematical Sciences*, 3(1):19, 2016.

[14] M.W.M.G. Dissanayake and N. Phan-Thien. Neural network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10:195–201, 1994.

[15] S. Dong and Z. Li. Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 387:114129, 2021. (also arXiv:2012.02895).

[16] S. Dong and Z. Li. A modified batch intrinsic plascity method for pre-training the random coefficients of extreme learning machines. *Journal of Computational Physics*, 445:110585, 2021. (also arXiv:2103.08042).

[17] S. Dong and N. Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435:110242, 2021.

[18] S. Dong and J. Shen. A pressure correction scheme for generalized form of energy-stable open boundary conditions for incompressible flows. *Journal of Computational Physics*, 291:254–278, 2015.

[19] S. Dong and Y. Wang. A method for computing inverse parametric pde problems with random-weight neural networks. *Journal of Computational Physics*, 489:112263, 2023. (also arXiv:2210.04338).

[20] S. Dong and J. Yang. Numerical approximation of partial differential equations by a variable projection

method with artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, 398:115284, 2022. (also arXiv:2201.09989).

[21] S. Dong and J. Yang. On computing the hyperparameter of extreme learning machines: algorithms and applications to computational PDEs, and comparison with classical and high-order finite elements. *Journal of Computational Physics*, 463:111290, 2022. (also arXiv:2110.14121).

[22] V. Dwivedi and B. Srinivasan. Physics informed extreme learning machine (pielm) − a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 391:96–118, 2020.

[23] V. Dwivedi and B. Srinivasan. A normal equation-based extreme learning machine for solving linear partial differential equations. *Journal of Computing and Information Science in Engineering*, 22:014502, 2022.

[24] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.*, 5:349380, 2017.

[25] W. E and B. Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6:1–12, 2018.

[26] G. Fabiani, F. Calabro, L. Russo, and C. Siettos. Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines. *Journal of Scientific Computing*, 89:44, 2021.

[27] G. Fabiani, E. Galaris, L. Russo, and C. Siettos. Parsimonious physics-informed random projection neural networks for initial value problems of ODEs and index-1 DAEs. *Chaos*, 33:043128, 2023.

[28] M. De Florio, E. Schiassi, F. Calabro, and R. Furfaro. Physics-informed neural networks for 2nd order ODEs with sharp gradients. *Journal of Computational and Applied Mathematics*, 436:115396, 2023.

[29] A.L. Freire, A.R. Rocha-Neto, and G.A. Barreto. On robust randomized neural networks for regression: a comprehensive review and evaluation. *Neural Computing and Applications*, 32:16931–16950, 2020.

[30] L. Gonon. Random feature neural networks learn Black-Scholes type PDEs without curse of dimensionality. *Journal of Machine Learning Research*, 24:1–51, 2023.

[31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[32] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of National Academy of Sciences of USA*, 115(34):8505–8510, 2018.

[33] Jihun Han, Mihai Nica, and Adam R Stinchcombe. A derivative-free method for solving elliptic partial differential equations with deep neural networks. *Journal of Computational Physics*, 419:109672, 2020.

[34] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[35] G. Huang, G.B. Huang, S. Song, and K. You. Trends in extreme learning machines: a review. *Neural Networks*, 61:32–48, 2015.

[36] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks*, volume 2, pages 985–990, 2004.

[37] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70:489–501, 2006.

[38] G.B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17:879–892, 2006.

[39] M. Hutzenthaler, A. Jentzen, T. Kruse, and T.A. Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *Partial Differ. Equ. Appl.*, 1:34, 2020.

[40] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, et al. On multilevel picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *Journal of Scientific Computing*, 79(3):1534–1571, 2019.

[41] B. Igelnik and Y.H. Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6:1320–1329, 1995.

[42] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20:335–352, 2007.

[43] A.D. Jagtap and G.E. Karniadakis. Extended physics-informed neural network (XPINNs): A general-

ized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28:2002–2041, 2020.

[44] A.D. Jagtap, E. Kharazmi, and G.E. Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[45] A. Jentzen, D. Salimova, and T. Welti. A proof that artificial neural networks overcomes the curse of dimensionality in the numerical approximation of kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *Commun. Math. Sci.*, 19:1167–1205, 2021.

[46] G.E. Karniadakis, G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021.

[47] G.E. Karniadakis and S.J. Sherwin. *Spectral/hp element methods for computational fluid dynamics, 2nd edn.* Oxford University Press, 2005.

[48] A.S. Krishnapriyan, A. Gholami, S. Zhe, R.M. Kirby, and M.W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *arXiv:2109.01050*, 2021.

[49] C. Leake, H. Johnston, and D. Mortari. *The Theory of Functional Connections: A Functional Interpolation Framework with Applications.* Lulu, 2022.

[50] H. Lee and I. Kang. Neural algorithms for solving differential equations. *Journal of Computational Physics*, 91:110–117, 1990.

[51] J.-Y. Li, W. Chow, B. Igelnik, and Y.-H. Pao. Comments on "stochastic choice of basis functions in adaptive function approximaton and the functional-link net". *IEEE Trans. Neural Netw.*, 8:452–454, 1997.

[52] K. Li, K. Tang, T. Wu, and Q. Liao. D3M: A deep domain decomposition method for partial differential equations. *IEEE Access*, 8:5283–5294, 2020.

[53] M. Li and D. Wang. Insights into randomized algorithms for neural networks: practical issues and common pitfalls. *Information Sciences*, 382–383:170–178, 2017.

[54] S. Li, G. Liu, and S. Xiao. Extreme learning machine with kernels for solving elliptic partial differential equations. *Cognitive Computing*, 15:413–428, 2023.

[55] Y. Liao and P. Wang. Deep Nitsche method: deep Ritz method with essential boundary conditions. *Commun. Comput. Phys.*, 29:1365–1384, 2021.

[56] L. Lin, Z. Yang, and S. Dong. Numerical approximation of incompressible Navier-Stokes equations based on an auxiliary energy variable. *Journal of Computational Physics*, 388:1–22, 2019.

[57] H. Liu, B. Xing, Z. Wang, and L. Li. Legendre neural network method for several classes of singularly perturbed differential equations based on mapping and piecewise optimization technology. *Neural Processing Letters*, 51:2891–2913, 2020.

[58] M. Liu, M. Hou, J. Wang, and Y. Cheng. Solving two-dimensional linear partial differential equations based on Chebyshev neural network with extreme learning machine algorithm. *Engineering Computations*, 38:874–894, 2021.

[59] L. Lu, H. Guo, X. Yang, and Y. Zhu. Temporal difference learning for high-dimensional PIDEs with jumps. *arXiv:2307.02766*, 2023.

[60] 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.

[61] Yulong Lu, Jianfeng Lu, and Min Wang. A priori generalization analysis of the deep ritz method for solving high dimensional elliptic partial differential equations. In *Conference on learning theory*, pages 3196–3241. PMLR, 2021.

[62] M. Lukosevicius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3:127–149, 2009.

[63] L. Lyu, Z. Zhang, M. Chen, and J. Chen. MIM: a deep mixed residual method for solving high-order particle differential equations. *Journal of Computational Physics*, 452:110930, 2022.

[64] A.J. Meade and A.A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Math. Comput. Modeling*, 19(12):1–25, 1994.

[65] D. Mortari. The theory of connections: connecting points. *Mathematics*, 5:57, 2017.

[66] D. Mortari and C. Leake. The multivariate theory of connections. *Mathematics*, 7:296, 2019.

[67] Mohammad Amin Nabian and Hadi Meidani. A deep learning solution approach for high-dimensional random differential equations. *Probabilistic Engineering Mechanics*, 57:14–25, 2019.

[68] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high-dimensional hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021.
[69] D. Needell, A.A. Nelson, R. Saab, and P. Salanevich. Random vector functional link networks for function approximation on manifolds. *arXiv:2007.15776*, 2020.
[70] N. Ni and S. Dong. Numerical computation of partial differential equations by hidden-layer concatenated extreme learning machine. *Journal of Scientific Computing*, 95:35, 2023. (also arXiv:2204.11375).
[71] S. Panghal and M. Kumar. Optimization free neural network approach for solving ordinary and partial differential equations. *Engineering with Computers*, 37:2989–3002, 2021.
[72] Y.H. Pao, G.H. Park, and D.J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6:163–180, 1994.
[73] Y.H. Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25:76–79, 1992.
[74] M. Penwarden, A.D. Jagtap, S. Zhe, G.E. Karniadakis, and R.M. Kirby. A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions. *Journal of Computational Physics*, in press, 2023. DOI: https://doi.org/10.1016/j.jcp.2023.112464.
[75] H.D. Quan and H.T. Huynh. Solving partial differential equation based on extreme learning machine. *Mathematics and Computers in Simulations*, 205:697–708, 2023.
[76] A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. *In D. Koller, D. Schuurmans, Y. Bengio and L. Bottou, editors, Advances in Neural Information Processing Systems (NIPS)*, 2:1316–1323, 2008.
[77] M. Raissi. Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations. *arXiv:1804.07010*, 2018.
[78] 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.
[79] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65:386–408, 1958.
[80] L. Ruthotto, S.J. Osher, W. Li, L. Nurbekyan, and S.W. Fung. A machine learning framework for solving high-dimensional mean field game mean field control problems. *Proceedings of National Academy of Sciences of USA*, 117:9183–9193, 2020.
[81] S. Scardapane and D. Wang. Randomness in neural networks: an overview. *WIREs Data Mining Knowl. Discov.*, 7:e1200, 2017.
[82] E. Schiassi, M. De Florio, B.D. Ganapol, P. Picca, and R. Furfaro. Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics. *Annuals of Nuclear Energy*, 167:108833, 2022.
[83] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnson, and D. Mortari. Extreme theory of functional connections: a fast physics-informed neural network method for solving ordinary and partial differential equations. *Neurocomputing*, 457:334–356, 2021.
[84] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
[85] P.N. Suhanthan and R. Katuwal. On the origins of randomization-based feedforward neural networks. *Applied Soft Computing*, 105:107239, 2021.
[86] H. Sun, M. Hou, Y. Yang, T. Zhang, F. Weng, and F. Han. Solving partial differential equations based on bernsteirn neural network and extreme learning machine algorithm. *Neural Processing Letters*, 50:1153–1172, 2019.
[87] S. Wang, X. Yu, and P. Perdikaris. When and why PINNs fail to train: a neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.
[88] Y. Wang and G. Lin. Efficient deep learning techniques for multiphase flow simulation in heterogeneous porous media. *Journal of Computational Physics*, 401:108968, 2020.
[89] C.S. Webster. Alan Turing's unorganized machines and artificial neural networks: his remarkable early work and future possibilities. *Evol. Intel.*, 5:35–43, 2012.
[90] E Weinan, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional pdes: from nonlinear monte carlo to machine learning. *Nonlinearity*, 35(1):278, 2021.

[91] Y. Yang, M. Hou, and J. Luo. A novel improved extreme learning machine algorithm in solving ordinary differential equations by legendre neural network methods. *Advances in Differential Equations*, 469:1–24, 2018.

[92] Z. Yang and S. Dong. An unconditionally energy-stable scheme based on an implicit auxiliary energy variable for incompressible two-phase flows with different densities involving only precomputable coefficient matrices. *Journal of Computational Physics*, pages 229–257, 2019.

[93] H. You, Y. Yu, N. Trask, M. Gulian, and M. D'Elia. Data-driven learning of nonlocal physics from high-fidelity synthetic data. *Computer Methods in Applied Mechanics and Engineering*, 374:113553, 2021.

[94] Y. Yu, R.M. Kirby, and G.E. Karniadakis. Spectral element and hp methods. *Encyclopedia of Computational Mechanics, John Wiley and Sons, NY*, 1:1–43, 2017.

[95] S. Zeng, Y. Cai, and Q. Zou. Deep neural networks based temporal-difference methods for high-dimensional parabolic partial differential equations. *Journal of Computational Physics*, 468:111503, 2022.

[96] L. Zhang and P.N. Suganthan. A comprehensive evaluation of random vector functional link networks. *Inf. Sci.*, 367–368:1094–1105, 2016.

[97] X. Zheng and S. Dong. An eigen-based high-order expansion basis for structured spectral elements. *Journal of Computational Physics*, 230:8573–8602, 2011.

[98] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.