Journal of Scientific Computing manuscript No.
(will be inserted by the editor)

Numerical Computation of Partial Differential Equations by Hidden-Layer Concatenated Extreme Learning Machine

Naxian Ni · Suchuan Dong

Received: date / Accepted: date

Abstract Extreme learning machine (ELM) is a type of randomized neural networks originally developed for linear classification and regression problems in the mid-2000s, and has recently been extended to computational partial differential equations (PDE). This method can yield highly accurate solutions to linear/nonlinear PDEs, but requires the last hidden layer of the neural network to be wide to achieve a high accuracy. If the last hidden layer is narrow, the accuracy of the existing ELM method will be poor, irrespective of the rest of the network configuration. In this paper we present a modified ELM method, termed HLConcELM (hidden-layer concatenated ELM), to overcome the above drawback of the conventional ELM method. The HLConcELM method can produce highly accurate solutions to linear/nonlinear PDEs when the last hidden layer of the network is narrow and when it is wide. The new method is based on a type of modified feedforward neural networks (FNN), termed HLConcFNN (hidden-layer concatenated FNN), which incorporates a logical concatenation of the hidden layers in the network and exposes all the hidden nodes to the output-layer nodes. HLConcFNNs have the interesting property that, given a network architecture, when additional hidden layers are appended to the network or when extra nodes are added to the existing hidden layers, the representation capacity of the HLConcFNN associated with the new architecture is guaranteed to be not smaller than that of the original network architecture. Here representation capacity refers to the set of all functions that can be exactly represented by the neural network of a given architecture. We present ample benchmark tests with linear/nonlinear PDEs to demonstrate the computational accuracy and performance of the HLConcELM method and the superiority of this method to the conventional ELM from previous works.

Keywords extreme learning machine · hidden layer concatenation · random weight neural networks · least squares · scientific machine learning · random basis

Naxian Ni
Center for Computational and Applied Mathematics, Department of Mathematics, Purdue University, USA. E-mail: nin@purdue.edu

Suchuan Dong
Center for Computational and Applied Mathematics, Department of Mathematics, Purdue University, USA. E-mail: sdong@purdue.edu

Mathematics Subject Classification (2020) 65D15 · 65M70 · 65M99 · 65N99 · 68T07

1 Introduction

This work extends our recent studies [9,10,13] of a type of random-weight neural networks, the so-called extreme learning machines (ELMs) [30], for scientific computing and in particular for computational partial differential equations (PDEs). Specifically, we would like to address the following question:

– Can ELM achieve a high accuracy for solving linear/nonlinear PDEs on network architectures with a narrow last hidden layer?

For the existing ELM method [9,10,13] (referred to as the conventional ELM hereafter), the answer to this question is negative. The goal of this paper is to introduce a modified method, referred to as the hidden-layer concatenated ELM (HLConcELM), to overcome this drawback of the conventional ELM and provide a positive answer to the above question.

Exploiting randomization in neural networks has a long history [53]. Turing's un-organized machine [63] and Rosenblatt's perceptron [52] in the 1950s are early examples of randomized neural networks. After a hiatus of several decades, there has been a strong revival of methods based on random-weight neural networks, starting in the 1990s [56]. In recent years randomization based neural networks have attracted a growing interest in a variety of areas [53,20].

Since it is enormously costly and hard to optimize the entire set of adjustable parameters in the neural network, it seems advisable if one randomly assigns and fixes a subset of the network's parameters so that the ensuing optimization task of network training can be simpler, and ideally linear, without severely compromising the network's achievable approximation capability. This strategy underlies the randomization of neural networks. When applied to feedforward or recurrent neural networks, randomization leads to techniques such as the random vector functional link (RVFL) networks [49,48,31], the extreme learning machine [29,30,26], the echo-state network [32,43], the no-propagation network [64], and the liquid state machine [44]. The random-weight neural networks (with a single hidden layer) are universal function approximators. The universal approximation property of such networks has been studied in [31,39,26,45]. The theoretical results of [31,26,45] establish that a single hidden-layer feedforward neural network having random but fixed (not trained) hidden nodes can approximate any continuous function to any desired degree of accuracy, provided that the number of hidden nodes is sufficiently large. The expected rate of convergence in the approximation of Lipschitz continuous functions is given in [31,50,45].

ELM was originally developed in [29,30] for single hidden-layer feedforward neural networks for linear classification/regression problems. It has since undergone a dramatic growth and found widespread applications in a variety of areas (see e.g. the reviews of [27,1] and the references therein). The method is based on two strategies: (i) randomly assigned but fixed (not trainable) hidden-layer coefficients, and (ii) trainable linear output-layer coefficients computed by a linear least squares method or by using the pseudoinverse (Moore-Penrose inverse) of the coefficient matrix [59,48,3,23].

While ELM emerged nearly two decades ago, the investigation of this technique for the numerical solution of differential equations has appeared only quite recently, alongside the proliferation of deep neural network (DNN) based PDE solvers in the past few years (see e.g. [54,51,17,66,24,8,33,62,11,42,34,58,60,37,61], among many others). In [67,57,40,41] the ELM technique has been used for solving linear ordinary or partial differential equations (ODEs/PDEs) with single hidden-layer feedforward neural networks, in which certain polynomials (e.g. Legendre, Chebyshev, or Bernstein polynomials) serve as the activation function. In [47] the ELM algorithm is used for solving linear ODEs and PDEs on neural networks with a single hidden layer, in which the Moore-Penrose inverse of the coefficient matrix has been used. In [15] a physics-informed ELM method is proposed for solving linear PDEs by combining the physics-informed neural network and the ELM idea. The neural network consists of a single hidden layer, and the Moore-Penrose inverse is employed to solve the resultant linear system. Interestingly, the authors set the number of hidden nodes to be equal to the total number of conditions in the problem. A solution strategy based on the normal equation associated with the linear system is studied in [16].

The ELM approach is extended to the numerical solution of nonlinear PDEs in [9] on local or global feedforward neural networks with a single or multiple hidden layers. A nonlinear least squares method with perturbations (NLLSQ-perturb) and a Newton-linear least squares (Newton-LLSQ) method are developed for solving the resultant nonlinear algebraic system for the output-layer coefficients of the ELM neural network. The NLLSQ-perturb algorithm therein takes advantage of the nonlinear least squares implementation from the scipy library, which implements a Gauss-Newton type method combined with a trust-region strategy. A block time marching (BTM) scheme is proposed in [9] for long-time dynamic simulations of linear and nonlinear PDE problems, in which the temporal dimension (if large) is divided into a number of windows (called time blocks) and the PDE problem is solved on the time blocks individually and successively. More importantly, a systematic comparison of the accuracy and the computational cost (network training time) between the ELM method and two state-of-the-art deep neural network (DNN) based PDE solvers, the deep Galerkin method (DGM) [54] and the physics-informed neural network (PINN) method [51], has been conducted in [9], as well as a systematic comparison between ELM and the classical finite element method (FEM). The comparisons show that the ELM method far outperforms DGM and PINN in terms of the accuracy and the computational cost, and that ELM is on par with the classical FEM in computational performance and outperforms the FEM as the problem size becomes larger. In [9] the hidden-layer coefficients are set to uniform random values generated on $[-R_m, R_m]$, where R_m is a user-prescribed constant. The results of [9] show that the R_m value has a strong influence on the numerical accuracy of the ELM results and that the best accuracy is associated with a range of moderate R_m values for a given problem. This is consistent with the observation for classification problems [71].

A number of further developments of the ELM technique for solving linear and nonlinear PDEs appeared recently; see e.g. [10,6,21,13,18], among others. In order to address the influence of random initialization of the hidden-layer coefficients on the ELM accuracy, a modified batch intrinsic plasticity (modBIP) method is developed in [10] for pre-training the random coefficients in the ELM network. This method, together with ELM, is applied to a number of linear and nonlinear

PDEs. The accuracy of the combined modBIP/ELM method has been shown to be insensitive to the random initializations of the hidden-layer coefficients. In [6] the authors have presented a method for solving one-dimensional linear elliptic PDEs based on ELM with single hidden-layer feedforward neural networks and the sigmoid activation function. The random parameters in the activation function are set based on the location of the domain of interest and the function derivative information. In [21] the authors present a method based on randomized neural networks with a single hidden layer for solving stiff ODEs. The time integration therein appears to be similar to the block time marching strategy [9], but with an adaptation on the time block sizes. It is observed that the presented method is advantageous over the stiff ODE solvers from MatLab. Noting the influence of the maximum random-coefficient magnitude (i.e. the R_m constant) on the ELM accuracy as shown by [9], in [13] we have presented a method for computing the optimal R_m in ELM based on the differential evolution algorithm, as well as an improved implementation for computing the differential operators of the last hidden-layer data. These improvements significantly enhance the ELM computational performance and dramatically reduce its network training time as compared with that of [9]. The improved ELM method is compared systematically with the traditional second-order and high-order finite element methods for solving a number of linear and nonlinear PDEs in [13]. The improved ELM far outperforms the second-order FEM. For smaller problem sizes it is comparable to the high-order FEM in performance, and for larger problem sizes the improved ELM outperforms the high-order FEM markedly. Here, by "outperform" we mean that one method achieves a better accuracy under the same computational cost or incurs a lower computational cost to achieve the same accuracy. In [18] an ELM method is presented for the numerical solution of stationary nonlinear PDEs based on the sigmoid and radial basis activation functions. The authors observe that the ELM method exhibits a better accuracy than the finite difference method and the FEM. Another recent development related to ELM is [12], in which a method based on the variable projection strategy is proposed for solving linear and nonlinear PDEs with artificial neural networks. For linear PDEs, the neural-network representation of the PDE solution leads to a separable nonlinear least squares problem, which is then reformulated to eliminate the output-layer coefficients, leading to a reduced problem about the hidden-layer coefficients only. The reduced problem is solved first by the nonlinear least squares method to determine the hidden-layer coefficients, and the output-layer coefficients are then computed by the linear least squares method [12]. For nonlinear PDEs, the problem is first linearized by the Newton's method with a particular linearization form, and the linearized system is solved by the variable projection framework together with neural networks. The ELM method can be considered as a special case of the variable projection, i.e. with zero iteration when solving the reduced problem for the hidden-layer coefficients [12]. It is shown in [12] that the variable projection method exhibits an accuracy significantly superior to the ELM under identical conditions and network configurations.

As has been shown in previous works [9, 10, 13], ELM can produce highly accurate results for solving linear and nonlinear PDEs. For smooth field solutions the ELM errors decrease exponentially as the number of training data points or the number of training parameters in the neural network increases, and the errors can reach a level close to the machine zero when the number of degrees of freedom becomes large. To achieve a high accuracy, however, the existing ELM method

requires the number of nodes in the last hidden layer of the neural network to be sufficiently large [9]. Therefore, the ELM network usually has a wide hidden layer in the case of a shallow neural network, or a wide last hidden layer in the case of deeper neural networks. If the last hidden layer contains only a small number of nodes, the results computed with the existing (conventional) ELM will tend to be poor in accuracy, regardless of the configuration with the rest of the network.

In this paper, we focus on feedforward neural networks (FNNs) with multiple hidden layers, and present a modified ELM method (termed HLConcELM) for solving PDEs to overcome the above drawback associated with conventional ELM. The HLConcELM method can produce accurate solutions to linear/nonlinear PDEs when the last hidden layer of the network is narrow, and when the last hidden layer is wide.

The new method is based on a type of modified feedforward neural networks, referred to as hidden-layer concatenated FNN (or HLConcFNN) herein, which incorporates a logical concatenation of the hidden layers so that all the hidden nodes are exposed to and connected with the nodes in the output layer (see Section 2 for details). The HLConcFNNs have the interesting property that, given a network architecture, when additional hidden layers are appended to the neural network or when extra nodes are added to the existing hidden layers, the representation capacity of the HLConcFNN associated with the new architecture is guaranteed to be not smaller than that associated with the original network architecture. Here by representation capacity we refer to the set of all functions that can be exactly represented by the neural network (see Section 2 for the definition). In contrast, conventional FNNs do not have a parallel property when additional hidden layers are appended to the network.

The HLConcELM is attained by assigning (and fixing) the weight/bias coefficients in the hidden layers of the HLConcFNN to random values, and allowing the connection coefficients between all the hidden nodes and the output nodes to be adjustable (trainable). More specifically, given a network architecture with L hidden layers, we set the weight/bias coefficients of the l-th ($1 \leqslant l \leqslant L$) hidden layer to uniform random values generated on the interval $[-R_l, R_l]$, where R_l is a constant. The vector of R_l constants (referred to as the hidden magnitude vector herein), $\mathbf{R} = (R_1, R_2, \ldots, R_L)$, influences the accuracy of HLConcELM, and in this paper we determine the optimal $\mathbf{R}$ using the method from [13] based on the differential evolution algorithm. HLConcELMs partially inherit the non-decreasing representation capacity property of HLConcFNNs. For example, given a network architecture, when extra hidden layers are appended to the network, the representation capacity of the HLConcELM associated with the new architecture will not be smaller than that associated with the original architecture, provided that the random hidden-layer coefficients for the new architecture are assigned in an appropriate fashion. On the other hand, when extra nodes are added to the existing hidden layers, HLConcELMs in general do not have a parallel non-decreasing property with regard to its representation capacity, because of the randomly assigned hidden-layer coefficients.

The exploration of neural-network architecture has been actively pursued in machine learning research, and the connectivity patterns are the focus of a number of research efforts. Neural networks incorporating shortcut connections (concatenations) between the input nodes, the hidden nodes, and the output nodes are explored in e.g. [28,65,7,36,19] (among others). The hidden-layer concatenated

neural network adopted in the current paper can be considered in spirit as a simplification of the connection patterns in the DenseNet [28] architecture, and it is similar to the deep RVFL architecture of [36] but without the connection between the input nodes and the output nodes. We note that these previous works are for image and data classification problems, while the current work focuses on scientific computing and in particular the numerical solutions of partial differential equations.

We present extensive numerical experiments with linear and nonlinear PDEs to test the performance of the HLConcELM method and compare this method with the conventional ELM method. These benchmark tests demonstrate unequivocally that HLConcELM can achieve highly accurate results when the last hidden layer in the neural network is narrow or wide, and that it is much superior in accuracy to the conventional ELM. The implementation of the current method is in Python and employs the Tensorflow (www.tensorflow.org), Keras (keras.io), and the scipy libraries. All the benchmark tests are performed on a MAC computer (3.2GHz Intel Core i5 CPU, 24GB memory) in the authors' institution.

The contribution of this work lies in two aspects. The first one lies in the HLConcELM method for solving linear and nonlinear PDEs. The other aspect is with regard to the non-decreasing representation capacity of HLConcFNNs when additional hidden layers are appended to an existing network architecture. To the best the authors' knowledge, this property seems unknown to the community so far. Bringing this property into collective consciousness can be another contribution of this paper.

The rest of this paper is organized as follows. In Section 2 we discuss the structures of HLConcFNNs and HLConcELMs, as well as their non-decreasing representation capacity property when additional hidden layers are appended to an existing architecture. We then develop the algorithm for solving linear and nonlinear PDEs employing the HLConcELM architecture. In Section 3 we present extensive benchmark examples to test the current HLConcELM method and compare its performance with that of the conventional ELM. Section 4 concludes the presentation with several further comments about the presented method. In Appendix A we provide constructive proofs to the theorems from Section 2 concerning the representation capacity of HLConcFNNs and HLConcELMs. Appendix B summarizes a study of different activation functions with the HLConcELM method. Appendix C provides further comparisons between HLConcELM and conventional ELM under the setting that the number of trainable parameters is maintained to be the same in both methods. Further tests of the HLConcELM method are documented in Appendix D for the Laplace equation around a reentrant corner, in Appendix E for the Kuramoto-Sivashinsky equation, in Appendix F for the Shrodinger equation, and in Appendix G for the two-dimensional advection equation.

2 Hidden-Layer Concatenated Extreme Learning Machine

2.1 Conventional ELM and Drawback

The ELM method with feedforward neural networks (FNN) for solving linear and nonlinear PDEs has been described in e.g. [9,10,13]. Figure 1(a) illustrates such a network containing three hidden layers. From layer to layer, the arrow in the

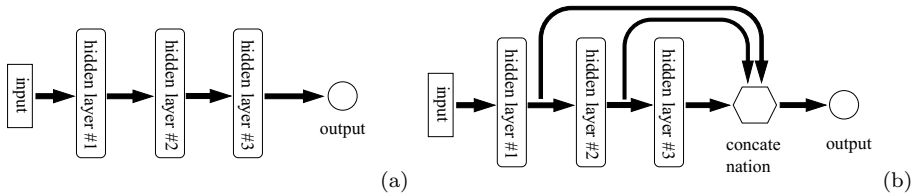

Fig. 1 Illustration of neural network structure (with 3 hidden layers): (a) conventional FNN, and (b) hidden-layer concatenated FNN (HLConcFNN). In HLConcFNN all the hidden nodes are exposed to the output nodes, while in conventional FNN only the last hidden-layer nodes are exposed to the output nodes.

sketch represents the usual FNN logic, an affine transform followed by a function composition with an activation function [22]. For ELM we require that no activation function be applied to the output layer and that the output layer contain no bias. So the output layer is linear and has zero bias. This requirement is adopted throughout this paper.

As discussed in [9], we pre-set the weight/bias coefficients in all the hidden layers to random values and fix these values once they are assigned. Only the weight coefficients of the output layer are trainable. The hidden-layer coefficients in the neural network are not trainable with ELM.

To solve a given linear or nonlinear PDE with ELM, we first enforce the PDE and the associated boundary/initial conditions on a set of collocation points in the domain or on the appropriate domain boundaries. This gives rise to a linear least squares problem for linear PDEs, or a nonlinear least squares problem for nonlinear PDEs, about the output-layer coefficients (trainable parameters) of the neural network [9]. We solve this least squares problem for the output-layer coefficients by a linear least squares method for linear PDEs and by a nonlinear least squares method for nonlinear PDEs [9].

ELM can produce highly accurate solutions to PDEs. In particular, for smooth solutions its errors decrease exponentially as the number of collocation points or the number of trainable parameters (the number of nodes in the last hidden layer) increases [9,10]. In addition, it has a low computational cost (network training time) [9,13].

Hereafter we refer to a vector or a list of positive integers as an architectural vector (denoted by $\mathbf{M}$),

$$\text{architectural vector:} \quad \mathbf{M} = [M_0, M_1, \ldots, M_{L-1}, M_L] \tag{1}$$

where $(L+1)$ is the dimension of the vector with $L \geqslant 2$, and M_i $(0 \leqslant i \leqslant L)$ are positive integers. We associate a given $\mathbf{M}$ to the architecture of an FNN with $(L+1)$ layers, where M_i $(0 \leqslant i \leqslant L)$ is the number of nodes in the i-th layer. The layer 0 and the layer L represent the input and the output layers, respectively. The layers in between are the hidden layers.

Despite its high accuracy and attractive computational performance, certain aspect of the ELM method is less appealing and remains to be improved. One particular aspect in this regard concerns the size of the last hidden layer of the ELM network. ELM requires the last hidden layer of the neural network to be wide in order to achieve a high accuracy, irrespective of the sizes of the rest of

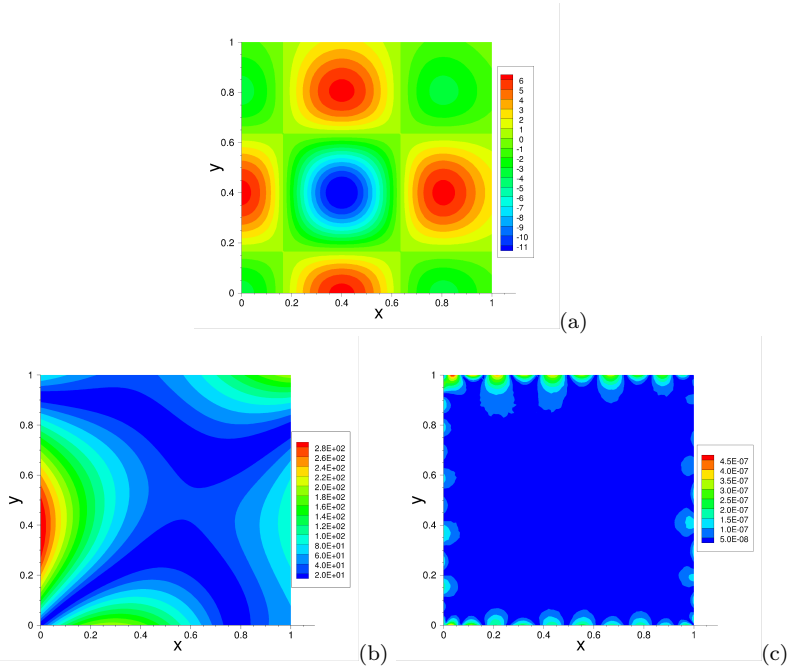

Fig. 2 Illustration of error characteristics of conventional ELM (Poisson equation): Distributions of (a) the exact solution, (b) the ELM error obtained with the architecture $\mathbf{M}_1 = [2, 300, 30, 1]$, and (c) the ELM error obtained with the architecture $\mathbf{M}_2 = [2, 300, 400, 1]$.

the network architecture. If the last hidden layer contains only a small number of nodes, the ELM accuracy will be poor even though the preceding hidden layers can be wide enough. This point is illustrated by Figure 2, which shows the ELM results for solving the two-dimensional (2D) Poisson equation on a unit square. Figure 2(a) shows the distribution of the exact solution. Figures 2(b) and (c) show the ELM error distributions obtained using two network architectures given by $[2, 300, 30, 1]$ and $[2, 300, 400, 1]$, respectively, under otherwise identical conditions. Both neural networks have the Gaussian activation function $\sigma(x) = e^{-x^2}$ for all the hidden nodes, and are trained on a uniform set of 21×21 collocation points. The only difference between them is the size of the last hidden layer. With 400 nodes in the last hidden layer the ELM solution is highly accurate, with the maximum error on the order 10^{-7} in the domain. With 30 nodes in the last hidden layer, on the other hand, the ELM solution exhibits no accuracy at all, with the maximum error on the order of 10^2, despite the fact that the first hidden layer is fairly large (containing 300 nodes). With the existing ELM method, only the last hidden-layer nodes directly contribute to the the output of the neural network, while the nodes in the preceding hidden layers do not directly affect the network output. In other words, with the existing ELM, all the degrees of freedom provided by the nodes in the preceding hidden layers are to some extent "wasted".

Can one achieve a high accuracy even if the last hidden layer is narrow in the ELM network? Can we take advantage of the degrees of freedom provided by the

hidden nodes in the preceding hidden layers with ELM? These are the questions we are interested in and would like to address in the current work.

The above drawback of the existing ELM method, which will be referred to as the conventional ELM hereafter, motivates the developments in what follows. We present a modified ELM method to address this issue and discuss how to use the modified method for numerical simulations of PDEs.

2.2 Modifying ELM Neural Network with Hidden-Layer Concatenation

To address the aforementioned drawback, we consider a type of modified FNNs for ELM computation. The idea of the modified network is illustrated in Figure 1(b) using three hidden layers as an example.

The main strategy here is to expose *all* the hidden nodes in the neural network to the output-layer nodes. Starting with a standard FNN, we incorporate a logical concatenation layer between the last hidden layer and the output layer. This logical layer concatenates the output fields of all the hidden nodes, from the first to the last hidden layers, in the original network architecture. From the logical concatenation layer to the output layer a usual affine transform, together with possibly a function composition with an activation function, is performed to attain the output fields of the overall neural network. Note that the logical concatenation layer involves no parameters.

Hereafter we refer to this type of modified neural networks as the hidden-layer concatenated FNN (HLConcFNN), and the original FNN as the base neural network. Thanks to the logical concatenation, in HLConcFNN all the hidden nodes in the base network architecture are connected with the output nodes.

One can also include the input fields in the logical concatenation layer. Numerical experiments show that, however, there is no advantage in terms of the accuracy when the input fields are included. In the current paper we do not include the input fields in the concatenation.

Let us next use a real-valued function of d ($d \geqslant 1$) variables, $u(\mathbf{x})$ ($\mathbf{x} \in \Omega \subset \mathbb{R}^d$), represented by a HLConcFNN to illustrate some of its properties. Consider a HLConcFNN, whose base architecture is given by $\mathbf{M}$ in (1), where $M_0 = d$ and $M_L = 1$. The d input nodes represent the d components of $\mathbf{x} = (x_1, x_2, \ldots, x_d)$, and the single output node represents the function $u(\mathbf{x})$. Let $\sigma : \mathbb{R} \to \mathbb{R}$ denote the activation function for all the hidden nodes. As stated before, we require that no activation function be applied to the output node and that it contain no bias.

Let $\boldsymbol{\Phi}^{(i)}(\mathbf{x}) = \left(\phi_1^{(i)}(\mathbf{x}), \ldots, \phi_{M_i}^{(i)}(\mathbf{x})\right)$, $1 \leqslant i \leqslant L-1$, denote the M_i output fields of the i-th hidden layer. The logical concatenation layer contains a total of $N_c(\mathbf{M}) = \sum_{i=1}^{L-1} M_i$ logical nodes. Then we have the following expansion relation,

$$u(\mathbf{x}) = \sum_{i=1}^{L-1} \sum_{j=1}^{M_i} \beta_{ij} \phi_j^{(i)}(\mathbf{x}) = \sum_{i=1}^{L-1} \boldsymbol{\Phi}^{(i)}(\mathbf{x}) \boldsymbol{\beta}_i^T = \boldsymbol{\Phi}(\mathbf{x}) \boldsymbol{\beta}^T, \tag{2}$$

where β_{ij} ($1 \leqslant i \leqslant L-1$, $1 \leqslant j \leqslant M_i$) denotes the weight coefficient of the output layer, i.e. the connection coefficient between the output node and the j-th hidden

node in the i-th hidden layer, and

$$\begin{cases} \boldsymbol{\beta}_i = (\beta_{i1}, \beta_{i2}, \ldots, \beta_{iM_i}), \\ \boldsymbol{\beta} = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_{L-1}) = (\beta_{11}, \ldots, \beta_{1M_1}, \beta_{21}, \ldots, \beta_{L-1,M_{L-1}}), \\ \boldsymbol{\Phi} = \left(\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \ldots, \boldsymbol{\Phi}^{(L-1)}\right) = \left(\phi_1^{(1)}, \ldots, \phi_{M_1}^{(1)}, \phi_1^{(2)}, \ldots, \phi_{M_{L-1}}^{(L-1)}\right). \end{cases} \tag{3}$$

The logic from layer $(i-1)$ to layer i, for $1 \leqslant i \leqslant L-1$, represents an affine transform followed by a function composition with the activation function,

$$\begin{cases} \phi_j^{(i)}(\mathbf{x}) = \sigma\left(\sum_{k=1}^{M_{i-1}} \phi_k^{(i-1)}(\mathbf{x}) w_{kj}^{(i)} + b_j^{(i)}\right), \quad 1 \leqslant j \leqslant M_i, \\ \boldsymbol{\Phi}^{(i)}(\mathbf{x}) = \sigma\left(\boldsymbol{\Phi}^{(i-1)}(\mathbf{x})\mathbf{W}^{(i)} + \mathbf{b}^{(i)}\right). \end{cases} \tag{4}$$

In the above equation, the constants $w_{kj}^{(i)}$ $(1 \leqslant k \leqslant M_{i-1},\ 1 \leqslant j \leqslant M_i)$ are the weights and $b_j^{(i)}$ $(1 \leqslant j \leqslant M_i)$ are the biases of layer i, and

$$\begin{cases} \mathbf{W}^{(i)} = \left[w_{kj}^{(i)}\right]_{M_{i-1} \times M_i}, \quad \mathbf{b}^{(i)} = \left(b_1^{(i)}, b_2^{(i)}, \ldots, b_{M_i}^{(i)}\right), \\ \boldsymbol{\Phi}^{(0)}(\mathbf{x}) = \left(\phi_1^{(0)}, \phi_2^{(0)}, \ldots, \phi_{M_0}^{(0)}\right) = \mathbf{x}. \end{cases} \tag{5}$$

Define

$$\begin{cases} \boldsymbol{\theta}^{(i)} = \mathbf{flatten}\left[\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\right], \quad 1 \leqslant i \leqslant L-1, \\ \boldsymbol{\theta} = (\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L-1)}) = (\theta_1, \theta_2, \ldots, \theta_{N_h}), \end{cases} \tag{6}$$

where "**flatten**" denotes the operation to reshape and combine a list of matrices or vectors into a single vector, and $N_h(\mathbf{M}) = \sum_{i=1}^{L-1}(M_{i-1}+1)M_i$. Here $\boldsymbol{\theta}^{(i)}$ denotes the vector of weight/bias coefficients of layer i for $1 \leqslant i \leqslant L-1$, $\boldsymbol{\theta}$ denotes the vector of weight/bias coefficients in all the hidden layers, and N_h is the total number of hidden-layer coefficients in the neural network.

The output field of the neural network depends on $(\boldsymbol{\theta}, \boldsymbol{\beta})$, and the output fields of each hidden layer depend on $\boldsymbol{\theta}$. To make these dependencies more explicit, we re-write equation (2) into

$$u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x}) = \sum_{i=1}^{L-1} \sum_{j=1}^{M_i} \beta_{ij} \phi_j^{(i)}(\boldsymbol{\theta}, \mathbf{x}) = \sum_{i=1}^{L-1} \boldsymbol{\Phi}^{(i)}(\boldsymbol{\theta}, \mathbf{x}) \boldsymbol{\beta}_i^T = \boldsymbol{\Phi}(\boldsymbol{\theta}, \mathbf{x}) \boldsymbol{\beta}^T, \tag{7}$$

where $\boldsymbol{\Phi}$, $\boldsymbol{\Phi}^{(i)}$, $\boldsymbol{\beta}_i$ and $\boldsymbol{\beta}$ are defined in (3).

A hidden-layer concatenated FNN is characterized by the architectural vector of the base network and the activation function. Given the architectural vector $\mathbf{M}$ and an activation function σ, let HLConcFNN$(\mathbf{M}, \sigma)$ denote the associated hidden-layer concatenated neural network. For a given domain $\Omega \subset \mathbb{R}^d$, an architectural vector $\mathbf{M} = (M_0, M_1, \ldots, M_L)$ with $M_0 = d$ and $M_L = 1$, and an activation function $\sigma(\cdot)$, we define

$$U(\Omega, \mathbf{M}, \sigma) = \{u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x}) \mid u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x}) \text{ is the output of HLConcFNN}(\mathbf{M}, \sigma), \\ \mathbf{x} \in \Omega,\ \boldsymbol{\theta} \in \mathbb{R}^{N_h},\ \boldsymbol{\beta} \in \mathbb{R}^{N_c}\} \tag{8}$$

as the collection of all possible output fields of this HLConcFNN$(\mathbf{M}, \sigma)$. $U(\Omega, \mathbf{M}, \sigma)$ denotes the set of all functions that can be exactly represented by this HLConcFNN$(\mathbf{M}, \sigma)$ on Ω. Hereafter we refer to $U(\Omega, \mathbf{M}, \sigma)$ as the representation capacity of the HLConcFNN$(\mathbf{M}, \sigma)$ for the domain Ω.

Remark 1 It should be noted that $U(\Omega, \mathbf{M}, \sigma)$ as defined by (8) is not a linear space, for the simple fact that it is not closed under addition because of the nonlinear parameters $\boldsymbol{\theta}$.

The HLConcFNNs have an interesting property. If one appends extra hidden layers to the network architecture, or adds nodes to any of the existing hidden layers, the representation capacity of the resultant HLConcFNN is at least as large as that of the original one. On the other hand, conventional FNNs lack such a property when additional hidden layers are appended to the architecture. Specifically, we have the following results.

Theorem 1 *Given an architectural vector* $\mathbf{M}_1 = (m_0, m_1, \ldots, m_{L-1}, m_L)$ *with* $m_L = 1$, *define a new vector* $\mathbf{M}_2 = (m_0, m_1, \ldots, m_{L-1}, n, m_L)$, *where* $n \geqslant 1$ *is an integer. For a given domain* $\Omega \subset \mathbb{R}^{m_0}$ *and an activation function* $\sigma(\cdot)$, *the following relation holds*

$$U(\Omega, \mathbf{M}_1, \sigma) \subseteq U(\Omega, \mathbf{M}_2, \sigma), \tag{9}$$

where U *is defined in* (8).

Theorem 2 *Given an architectural vector* $\mathbf{M}_1 = (m_0, m_1, \ldots, m_L)$ *with* $m_L = 1$, *define a new vector* $\mathbf{M}_2 = (m_0, m_1, \ldots, m_{s-1}, m_s + 1, m_{s+1}, \ldots, m_L)$ *for some* s *(*$1 \leqslant s \leqslant L-1$*). For a given domain* $\Omega \subset \mathbb{R}^{m_0}$ *and an activation function* $\sigma(\cdot)$, *the following relation holds*

$$U(\Omega, \mathbf{M}_1, \sigma) \subseteq U(\Omega, \mathbf{M}_2, \sigma), \tag{10}$$

where U *is defined in* (8).

These properties can be shown to be true by simple constructions, which are quite straightforward to devise. Risking on the side of naivety, we still include the constructive proofs for these two theorems in an Appendix of this paper for the benefit of a skeptical reader.

It should be noted that for conventional FNNs the relation given by (10) is true, but the relation given by (9) does not hold. Relation (9) is true for HLConcFNNs thanks to the concatenation of hidden layers in such networks.

Suppose we start with a base neural network architecture $\mathbf{M}_0$ and generate a sequence of architectures $\mathbf{M}_i$ $(i \geqslant 1)$, with each one obtained either by adding extra nodes to the existing hidden layers of or by appending additional hidden layers to the previous architecture. Then based on the above two theorems the HLConcFNNs associated with this sequence of architectures exhibit a hierarchical structure, in the sense that the representation capacities of this sequence of HLConcFNNs do not decrease, namely

$$U(\Omega, \mathbf{M}_0, \sigma) \subseteq U(\Omega, \mathbf{M}_1, \sigma) \subseteq \cdots \subseteq U(\Omega, \mathbf{M}_n, \sigma) \subseteq \cdots. \tag{11}$$

If the activation function $\sigma(\cdot)$ is nonlinear, the representation capacities of this sequence of HLConcFNNs should strictly increase.

Remark 2 If the number of nodes in the output layer of the HLConcFNN is more than one, the relations given by Theorems 1 and 2 about the representation capacities equally hold.

Hidden-Layer Concatenated Extreme Learning Machine (HLConcELM) Let us next combine the hidden-layer concatenated FNN with the idea of ELM. We adopt HLConcFNNs as the neural network for the ELM computation. We pre-set (and fix) all the weight/bias coefficients in the hidden layers (i.e. $\boldsymbol{\theta}$) of the HLConcFNN to random values, and train/compute the output-layer coefficients (i.e. $\boldsymbol{\beta}$) by a linear or nonlinear least squares method. We will refer to the resultant method as the hidden-layer concatenated extreme learning machine (HLConcELM).

Given an architectural vector $\mathbf{M}$, an activation function $\sigma(\cdot)$, and the randomly assigned values for the hidden-layer coefficients $\boldsymbol{\theta}$, let HLConcELM$(\mathbf{M}, \sigma, \boldsymbol{\theta})$ denote the associated hidden-layer concatenated ELM. For a given domain $\Omega \subset \mathbb{R}^d$, a vector $\mathbf{M} = (M_0, M_1, \ldots, M_L)$ with $M_0 = d$ and $M_L = 1$, and given $\boldsymbol{\theta} \in \mathbb{R}^{N_h}$ and σ, we define

$$U(\Omega, \mathbf{M}, \sigma, \boldsymbol{\theta}) = \{u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x}) \mid u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x}) \text{ is the output of HLConcFNN}(\mathbf{M}, \sigma), \\ \mathbf{x} \in \Omega,\ \boldsymbol{\beta} \in \mathbb{R}^{N_c}\} \quad (12)$$

as the set of all possible output fields of HLConcELM$(\mathbf{M}, \sigma, \boldsymbol{\theta})$ on Ω, where N_c denotes the total number of the output-layer coefficients. Hereafter we refer to $U(\Omega, \mathbf{M}, \sigma, \boldsymbol{\theta})$ as the representation capacity of the HLConcELM$(\mathbf{M}, \sigma, \boldsymbol{\theta})$. Note that $U(\Omega, \mathbf{M}, \sigma, \boldsymbol{\theta})$ forms a linear space.

Analogous to Theorem 1, when one appends hidden layers to a given network architecture, the representation capacity of the HLConcELM associated with the resultant architecture will be at least as good as that associated with the original one, on condition that the random hidden-layer coefficients of the new HLConcELM are set appropriately. On the other hand, if one adds extra nodes to a hidden layer (other than the last one) of a given architecture, there is no analogous result to Theorem 2 for HLConcELM, because the hidden-layer coefficients in ELM are randomly set. Specifically, we have the following result.

Theorem 3 *Given an architectural vector* $\mathbf{M}_1 = (m_0, m_1, \ldots, m_{L-1}, m_L)$ *with* $m_L = 1$, *define a new vector* $\mathbf{M}_2 = (m_0, m_1, \ldots, m_{L-1}, n, m_L)$, *where* $n \geqslant 1$ *is an integer. Let* $\boldsymbol{\theta} \in \mathbb{R}^{N_{h1}}$ *and* $\boldsymbol{\vartheta} \in \mathbb{R}^{N_{h2}}$ *denote two random vectors, with the relation* $\boldsymbol{\vartheta}[1 : N_{h1}] = \boldsymbol{\theta}[1 : N_{h1}]$, *where* $N_{h1} = \sum_{i=1}^{L-1}(m_{i-1} + 1)m_i$ *and* $N_{h2} = N_{h1} + (m_{L-1} + 1)n$. *For a given domain* $\Omega \subset \mathbb{R}^{m_0}$ *and an activation function* $\sigma(\cdot)$, *the following relation holds*

$$U(\Omega, \mathbf{M}_1, \sigma, \boldsymbol{\theta}) \subseteq U(\Omega, \mathbf{M}_2, \sigma, \boldsymbol{\vartheta}), \quad (13)$$

where U *is defined in* (12).

By $\boldsymbol{\vartheta}[1 : N_{h1}] = \boldsymbol{\theta}[1 : N_{h1}]$ we mean that the first N_{h1} entries of $\boldsymbol{\vartheta}$ and $\boldsymbol{\theta}$ are the same. Because of this condition, the random bases for $U(\Omega, \mathbf{M}_2, \sigma, \boldsymbol{\vartheta})$ would contain those bases for $U(\Omega, \mathbf{M}_1, \sigma, \boldsymbol{\theta})$, giving rise to the relation (13). For the sake of completeness we have included a proof of Theorem 3 in Appendix A. It should be noted that conventional ELMs lack a comparable property as expressed by the relation (13).

In the current paper we set the random hidden-layer coefficients $\boldsymbol{\theta}$ in HLConcELM in the following fashion. Given an architectural vector $\mathbf{M} = (m_0, m_1, \ldots, m_{L-1}, m_L)$, let $\boldsymbol{\xi} \in \mathbb{R}^{N_h}$ be a random vector generated on the interval $[-1, 1]$ from a uniform distribution, where $N_h = \sum_{i=1}^{L-1}(m_{i-1} + 1)m_i$. Once generated, $\boldsymbol{\xi}$ will be fixed throughout the computation for the given architecture $\mathbf{M}$. We next partition $\boldsymbol{\xi}$ into

$(L-1)$ sub-vectors, $\boldsymbol{\xi} = (\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_{L-1})$, with $\boldsymbol{\xi}_i$ having a dimension $(m_{i-1}+1)m_i$ for $1 \leqslant i \leqslant L-1$. Let $\boldsymbol{R} = (R_1, R_2, \ldots, R_{L-1})$ denote $(L-1)$ constants. We then set $\boldsymbol{\theta}$ in HLConcELM for the given architecture $\mathbf{M}$ to

$$\boldsymbol{\theta}(\mathbf{M}, \mathbf{R}, \boldsymbol{\xi}) = \mathbf{flatten}\left[R_1\boldsymbol{\xi}_1, R_2\boldsymbol{\xi}_2, \ldots, R_{L-1}\boldsymbol{\xi}_{L-1}\right], \tag{14}$$

where "**flatten**" concatenates the list of vectors into a single vector.

Hereafter we refer to the above vector $\mathbf{R} = (R_1, \ldots, R_{L-1})$ as the hidden magnitude vector for the network architecture $\mathbf{M}$. When assigning random hidden-layer coefficients as described above, we have essentially set the weight/bias coefficients in the i-th hidden layer to uniform random values generated on the interval $[-|R_i|, |R_i|]$, where R_i is the i-th component of $\mathbf{R}$, for $1 \leqslant i \leqslant L-1$. The constant $|R_i|$ denotes the maximum magnitude of the random coefficients for the i-th hidden layer.

The constants R_i $(1 \leqslant i \leqslant L-1)$ are the hyperparameters of the HLConcELM. The idea of generating random coefficients for different hidden layers with different maximum magnitudes is first studied in [13] for conventional feedforward neural networks, and a method based on the differential evolution algorithm is developed therein for computing the optimal values of those magnitudes. In the current work, for a given PDE problem, we use the method of [13] to compute the optimal (or near-optimal) hidden magnitude vector $\mathbf{R}^*$, and employ $\mathbf{R} = \mathbf{R}^*$ in HLConcELM for the simulations.

Hereafter we use HLConcELM$(\mathbf{M}, \sigma, \mathbf{R}, \boldsymbol{\xi})$ to denote the hidden-layer concatenated ELM characterized by the architectural vector $\mathbf{M}$, the activation function $\sigma(\cdot)$, the randomly-assigned but fixed vector $\boldsymbol{\xi}$ on $[-1, 1]$, and the hidden magnitude vector $\mathbf{R}$. According to Theorem 3, when additional hidden layers are appended to a given HLConcELM$(\mathbf{M}, \sigma, \mathbf{R}, \boldsymbol{\xi})$, the representation capacity of the resultant HLConcELM will not be smaller than that of the original one, if the vectors $\mathbf{R}$ and $\boldsymbol{\xi}$ of the resultant network are set appropriately.

2.3 Solving linear/nonlinear PDEs with Hidden-Layer Concatenated ELM

We next discuss how to use the hidden-layer concatenated ELM for the numerical solution of PDEs. Consider a domain $\Omega \subset \mathbb{R}^d$ and the following boundary value problem on this domain,

$$\mathcal{L}u + F(u) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{15a}$$

$$Bu + G(u) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega. \tag{15b}$$

In these equations $u(\mathbf{x})$ is the field function to be solved for. $\mathcal{L}$ is a linear differential operator. $F(u)$ is a nonlinear operator acting on u and also possibly on its derivatives. Equation (15b) represents the boundary conditions, where B is a linear differential or algebraic operator. The boundary condition may possibly contain some nonlinear operator $G(u)$ acting on u and also possibly on its derivatives. If both $F(u)$ and $G(u)$ are absent the problem becomes linear. We assume that this problem is well-posed.

In addition, we assume that $\mathcal{L}$ may possibly include time derivatives (e.g. $\frac{\partial}{\partial t}$, $\frac{\partial^2}{\partial t^2}$). In this case, problem (15) becomes time-dependent, and we will treat the time variable t in the same way as the spatial coordinate $\mathbf{x}$ and consider t as the

last dimension in d dimensions. We require that the equation (15b) should include appropriate initial condition(s) for such a case. So the problem (15) may refer to time-dependent cases, which will not be distinguished in the following discussions.

We represent the solution field $u(\mathbf{x})$ by a hidden-layer concatenated ELM from the previous subsection. Consider a network architecture given by $\mathbf{M} = (m_0, m_1, \ldots, m_L)$, where $m_0 = d$ and $m_L = 1$, and an activation function $\sigma(\cdot)$. We use the HLConcFNN$(\mathbf{M}, \sigma)$ to represent the solution field $u(\mathbf{x})$ (see Figure 1(b)). Here the d input nodes represent $\mathbf{x}$, and the single output node represents $u(\mathbf{x})$. The activation function σ is applied to all the hidden nodes in the network. As noted before, we require that the output layer should contain no activation function and have zero bias. With a given hidden magnitude vector $\mathbf{R} = (R_1, R_2, \ldots, R_{L-1})$ and a randomly generated vector $\boldsymbol{\xi} \in \mathbb{R}^{N_h}$ on $[-1, 1]$, where $N_h = \sum_{i=1}^{L-1}(m_{i-1}+1)m_i$, we set and fix the random hidden-layer coefficients according to equation (14).

Under these settings, the output field of the neural network is given by equation (2). Substituting this expression for $u(\mathbf{x})$ into the system (15), we have

$$\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\left[\mathcal{L}\phi_j^{(i)}(\mathbf{x})\right] + F\left(\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x})\right) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{16a}$$

$$\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\left[B\phi_j^{(i)}(\mathbf{x})\right] + G\left(\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x})\right) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \tag{16b}$$

where $\phi_j^{(i)}(\mathbf{x})$ $(1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i)$ denotes the output field of the j-th node in the i-th hidden layer, and β_{ij} $(1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i)$ are the weight coefficients in the output layer of the HLConcELM. It should be noted that, since the hidden-layer coefficients are randomly set but fixed, $\phi_j^{(i)}(\mathbf{x})$ are random but fixed functions. The coefficients β_{ij} are the trainable parameters in HLConcELM.

We next choose a set of Q $(Q \geqslant 1)$ points on Ω, referred to as the collocation points, which can be regular grid points, random points, or chosen based on some other distribution. Among these points we assume that Q_b $(1 \leqslant Q_b \leqslant Q-1)$ points reside on the boundary $\partial\Omega$ and the rest are from the interior of Ω. Let $\mathbb{X}$ denote the set of all the collocation points, and $\mathbb{X}_b$ denote the set of the boundary collocation points.

We enforce the equation (16a) on all the collocation points from $\mathbb{X}$, and enforce the equation (16b) on all the boundary collocation points from $\mathbb{X}_b$. This leads to

$$\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\left[\mathcal{L}\phi_j^{(i)}(\mathbf{x}_p)\right] + F\left(\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x}_p)\right) = f(\mathbf{x}_p), \quad \mathbf{x}_p \in \mathbb{X},\ 1 \leqslant p \leqslant Q; \tag{17a}$$

$$\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\left[B\phi_j^{(i)}(\mathbf{x}_q)\right] + G\left(\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x}_q)\right) = g(\mathbf{x}_q), \quad \mathbf{x}_q \in \mathbb{X}_b,\ 1 \leqslant q \leqslant Q_b. \tag{17b}$$

This is a system of $(Q + Q_b)$ nonlinear algebraic equations about $N_c = \sum_{i=1}^{L-1} m_i$ unknowns, β_{ij}. The differential operators involved in these equations, such as

$\mathcal{L}\phi_j^{(i)}(\mathbf{x}_p)$, $B\phi_j^{(i)}(\mathbf{x}_q)$, $F(u(\mathbf{x}_p))$ and $G(u(\mathbf{x}_q))$ where $u(\mathbf{x}_p) = \sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x}_p)$, can be computed by using automatic differentiation of the neural network.

The system (17) is a rectangular system, in which the number of equations and the number of unknowns are not the same. We seek a least squares solution to this system. This is a nonlinear least squares problem, and it can be solved by the Gauss-Newton method together with the trust region strategy [46]. Several quality implementations of the Gauss-Newton method are available from the scientific libraries. In this work we employ the Gauss-Newton implementation together with a trust region reflective algorithm [4,5] from the scipy package in Python (scipy.optimize.least_squares) to solve this problem. We refer to the method implemented in this scipy routine as the nonlinear least squares method in this paper.

The nonlinear least squares method requires two procedures for solving the system (17), one for computing the residual of this system and the other for computing the Jacobian matrix for a given arbitrary β_{ij} $(1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i)$. For a given arbitrary $\boldsymbol{\beta}$ (see (3)), the residual $\mathbf{r}(\boldsymbol{\beta}) \in \mathbb{R}^{Q+Q_b}$ is given by

$$\begin{cases} \mathbf{r}(\boldsymbol{\beta}) = (\mathbf{r}_1(\boldsymbol{\beta}), \mathbf{r}_2(\boldsymbol{\beta}))\,, \\ \mathbf{r}_1(\boldsymbol{\beta}) = (r_{11}, r_{12}, \ldots, r_{1Q}),\ \ \mathbf{r}_2(\boldsymbol{\beta}) = (r_{21}, r_{22}, \ldots, r_{2Q_b}); \\ r_{1p} = \displaystyle\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\left[\mathcal{L}\phi_j^{(i)}(\mathbf{x}_p)\right] + F\left(\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x}_p)\right) - f(\mathbf{x}_p), \\ \quad \mathbf{x}_p \in \mathbb{X}, \quad 1 \leqslant p \leqslant Q; \\ r_{2q} = \displaystyle\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\left[B\phi_j^{(i)}(\mathbf{x}_q)\right] + G\left(\sum_{i=1}^{L-1}\sum_{j=1}^{m_i}\beta_{ij}\phi_j^{(i)}(\mathbf{x}_q)\right) - g(\mathbf{x}_q), \\ \quad \mathbf{x}_q \in \mathbb{X}_b, \quad 1 \leqslant q \leqslant Q_b. \end{cases} \tag{18}$$

The Jacobian matrix $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\beta}} \in \mathbb{R}^{(Q+Q_b)\times N_c}$ is given by,

$$\begin{cases} \dfrac{\partial \mathbf{r}}{\partial \boldsymbol{\beta}} = \begin{bmatrix} \frac{\partial \mathbf{r}_1}{\partial \boldsymbol{\beta}} \\ \frac{\partial \mathbf{r}_2}{\partial \boldsymbol{\beta}} \end{bmatrix}_{(Q+Q_b)\times N_c}; \\ \dfrac{\partial \mathbf{r}_1}{\partial \boldsymbol{\beta}} = \left[\frac{\partial r_{1p}}{\partial \beta_{ij}}\right]_{Q\times N_c}, \quad \dfrac{\partial \mathbf{r}_2}{\partial \boldsymbol{\beta}} = \left[\frac{\partial r_{2q}}{\partial \beta_{ij}}\right]_{Q_b\times N_c}; \\ \dfrac{\partial r_{1p}}{\partial \beta_{ij}} = \mathcal{L}\phi_j^{(i)}(\mathbf{x}_p) + F'(u(\mathbf{x}_p))\phi_j^{(i)}(\mathbf{x}_p), \quad \mathbf{x}_p \in \mathbb{X}, \\ \quad 1 \leqslant p \leqslant Q, \quad 1 \leqslant i \leqslant L-1, \quad 1 \leqslant j \leqslant m_i; \\ \dfrac{\partial r_{2q}}{\partial \beta_{ij}} = B\phi_j^{(i)}(\mathbf{x}_q) + G'(u(\mathbf{x}_q))\phi_j^{(i)}(\mathbf{x}_q), \quad \mathbf{x}_q \in \mathbb{X}_b, \\ \quad 1 \leqslant q \leqslant Q_b, \quad 1 \leqslant i \leqslant L-1, \quad 1 \leqslant j \leqslant m_i; \end{cases} \tag{19}$$

where $u(\mathbf{x}_p)$ is computed based on equation (2). The $F'(u)$ and $G'(u)$ terms denote the derivatives with respect to u, and may represent the effect of an operator. For example, the nonlinear function $F(u) = u\frac{\partial u}{\partial x}$ (as in the Burgers' equation) leads to $F'(u)\phi = \frac{\partial u}{\partial x}\phi + u\frac{\partial \phi}{\partial x}$.

Therefore, to solve the problem (15) by HLConcELM, the input training data (denoted by $\mathbf{X}$) to the neural network is a $Q\times d$ matrix, consisting of the coordinates

of all the collocation points, $\mathbf{X} = \big[\mathbf{x}_p\big]_{Q\times d}$ (for all $\mathbf{x}_p \in \mathbb{X}$). The output data (denoted by $\mathbf{U}$) of the neural network is a $Q \times m_L$ matrix, representing the field solution $u(\mathbf{x})$ on the collocation points, $\mathbf{U} = \big[u(\mathbf{x}_p)\big]_{Q\times m_L}$. The output data of the logical concatenation layer (denoted by $\boldsymbol{\Psi}$) of the HLConcELM is a $Q \times N_c$ matrix given by $\boldsymbol{\Psi} = \big[\boldsymbol{\Phi}(\boldsymbol{\theta}, \mathbf{x}_p)\big]_{Q\times N_c}$. It represents the output fields of the all the hidden nodes on all the collocation points. Here N_c denotes the total number of hidden nodes in the network, and $\boldsymbol{\theta}$ denotes the random hidden-layer coefficients given by (14). The relation (7) is translated into, in terms of the neural-network data,

$$\mathbf{U} = \boldsymbol{\Psi}\boldsymbol{\beta}^T, \tag{20}$$

where $\boldsymbol{\beta}$ denotes the output-layer coefficients given by (3).

Remark 3 The output data of the logical concatenation layer $\boldsymbol{\Psi}$ can be computed by a forward evaluation of the neural network (for up to the logical concatenation layer) on the input data $\mathbf{X}$. In our implementation we have created a Keras sub-model with the input layer as its input and the logical concatenation layer as its output. By evaluating this sub-model on the input data we can attain the output data for all the hidden nodes on the collocation points. The first and higher derivatives of $\boldsymbol{\Psi}$ with respect to $\mathbf{X}$ are computed by a forward-mode auto-differentiation, implemented by the "ForwardAccumulator" in the Tensorflow library. This forward-mode auto-differentiation is crucial for the computational performance, because the total number of hidden nodes (N_c) in HLConcELM is typically much larger than the number of input nodes (d). The differential operators on the output fields of the hidden nodes involved in (18) and (19), such as $\mathcal{L}\phi_j^{(i)}(\mathbf{x}_p)$ ($\mathbf{x}_p \in \mathbb{X}$), $B\phi_j^{(i)}(\mathbf{x}_q)$ ($\mathbf{x}_q \in \mathbb{X}_b$) and $F'(u(\mathbf{x}_p))\phi_j^{(i)}(\mathbf{x}_p)$, can be computed based on or extracted from $\boldsymbol{\Psi}$ and its derivatives with respect to $\mathbf{X}$. Once $\boldsymbol{\Psi}$ is attained, for a given $\boldsymbol{\beta}$, the output data of the neural network can be computed by (20), which provides the $u(\mathbf{x}_p)$ ($\mathbf{x}_p \in \mathbb{X}$ or $\mathbf{x}_p \in \mathbb{X}_b$) for computing the terms $F'(u(\mathbf{x}_p))$ and $G'(u(\mathbf{x}_q))$ in (18) and (19).

Remark 4 If the boundary value problem (15) is linear, i.e. in the absence of the terms $F(u)$ and $G(u)$, the resultant system (17) is a linear algebraic system of $(Q+Q_b)$ equations about N_c unknowns of the parameters β_{ij}. In this case we use the linear least squares method to solve this system to compute a least squares solution for β_{ij}. In our implementation we employ the linear least squares routine from scipy (scipy.linalg.lstsq), which in turn employs the linear least squares implementation from the LAPACK library.

Remark 5 If the problem (15) is time-dependent, for longer-time or long-time simulations, we employ the block marching scheme from [9] together with the HLConcELM for its computation. The temporal dimension, which can be potentially large in this case, is first divided into a number of windows (referred as time blocks), so that each time block is of a moderate size. The problem on each time block is solved by HLConcELM individually and successively. After one time block is computed, the solution evaluated at the last time instant, possibly together with its derivatives, is used as the initial condition for computing the time block that follows. We refer the reader to [9] for more detailed discussions of the block time marching scheme.

Remark 6 HLConcFNNs can be used together with the locELM (local extreme learning machine) method [9] and domain decomposition for solving PDE problems. In this case, we employ a HLConcELM for the local neural network on each sub-domain, and the algorithm for computing the PDE solution is essentially the same. The only difference lies in that in the system (17) one needs to additionally include the C^k continuity conditions on those collocation points that reside on the sub-domain boundaries. The residuals in (18) and the Jacobian matrix in (19) need to be modified accordingly to account for these additional equations from the C^k continuity conditions. We refer the reader to [9] for detailed discussions of these aspects. For the convenience of presentation, hereafter we will refer to the locELM method based on HLConcFNNs as the locHLConcELM method (local hidden-layer concatenated ELM).

Remark 7 For a given problem, the optimal or near-optimal value $\mathbf{R}^*$ for the hidden magnitude vector $\mathbf{R}$ can be computed by the method from [13] based on the differential evolution algorithm. For all the test problems in Section 3, we employ $\mathbf{R} = \mathbf{R}^*$ computed based on the method of [13] in the HLConcELM simulations.

3 Numerical Benchmarks

In this section we employ several benchmark problems in two dimensions (2D) or in one spatial dimension (1D) plus time to test the performance of the HLConcELM method for solving linear and nonlinear PDEs. We show that this method can produce highly accurate results when the network architecture has a narrow last hidden layer. In contrast, the conventional ELM method in this case utterly loses accuracy.

The HLConcELM method is implemented in Python based on the Tensorflow and Keras libraries. The linear and nonlinear least squares methods employed in HLConcELM are based on the implementations in the scipy package (scipy.linalg.lstsq and scipy.optimize.least_squares), as discussed before. The differential operators on the hidden-layer data (see equations (17a)–(17b)) are computed by a forward-mode auto-differentiation, as stated in Remark 3. In all the numerical tests of this section we employ the Gaussian activation function $\sigma(x) = e^{-x^2}$ for all the hidden nodes, while the output layer is linear and has zero bias.

The ELM errors reported in the following subsections are computed as follows. We have considered regular rectangular domains for simplicity in the current paper. For a given architecture we train the HLConcELM network on $Q = Q_1 \times Q_1$ uniform collocation points (i.e. regular grid points) by the linear or nonlinear least squares method, with Q_1 uniform points in each direction of the 2D domain or the spatial-temporal domain. After the network is trained, we evaluate the neural network on a finer set of $Q_{eval} = Q_2 \times Q_2$ uniform grid points, with Q_2 much larger than Q_1, to attain the HLConcELM solution data. We evaluate the exact solution to the problem, if available, on the same set of Q_{eval} grid points. Then we compare the HLConcELM solution data and the exact solution data on the $Q_2 \times Q_2$ grid points to compute the maximum (l^∞) and root-mean-squares (rms, or l^2) errors. We refer to the errors computed above as the HLConcELM errors associated with the given network architecture and the $Q = Q_1 \times Q_1$ training collocation points. When Q_1 is varied in a range for the convergence tests, we have made sure that Q_2

is much larger than the largest Q_1 in the prescribed range. When the block time marching scheme is used for longer-time simulations together with HLConcELM (see Remark 5), the $Q = Q_1 \times Q_1$ and $Q_{eval} = Q_2 \times Q_2$ points above refer to the points in each time block. When the locHLConcELM method together with domain decomposition is used to solve a problem (see Remark 6), the Q and Q_{eval} points refer to the points in each sub-domain. In the current paper we employ a fixed $Q_{eval} = 101 \times 101$ (i.e. $Q_2 = 101$) when evaluating the neural network and computing the HLConcELM errors for all the test problems in this section.

As in our previous works [9,10], we employ a fixed seed for the random number generator in the Tensorflow library in order to make the reported numerical results herein exactly reproducible. While the seed value is different for the test problems in different subsections, it has been fixed to a particular value for the numerical tests within each subsection. Specifically, the seed to the random number generator is 10 in Section 3.1, 50 in Section 3.3, and 100 in Sections 3.2, 3.4 and 3.5.

In comparisons with the conventional ELM method [9] in the following subsections, all the hidden-layer coefficients in conventional ELM are assigned (and fixed) to uniform random values generated on the interval $[-R_m, R_m]$, with $R_m = R_{m0}$, where R_{m0} is the optimal R_m computed by the method of [13] based on the differential evolution algorithm.

3.1 Variable-Coefficient Poisson Equation

The first numerical test involves the 2D Poisson equation with a variable coefficient field. Consider the 2D domain $\Omega = [0, 1.6] \times [0, 1.6]$ and the the following boundary value problem on Ω,

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(a(x,y)\frac{\partial u}{\partial y}\right) = f(x,y), \tag{21a}$$

$$u(0,y) = g_1(y), \quad u(1.6,y) = g_2(y), \quad u(x,0) = h_1(x), \quad u(x,1.6) = h_2(x), \tag{21b}$$

where (x, y) are the spatial coordinates, $u(x, y)$ is the field function to be solved for, $f(x, y)$ is a prescribed source term, $a(x, y)$ is the coefficient field given by $a(x, y) = 2 + \sin(x + y)$, and g_1, g_2, h_1 and h_2 are prescribed boundary distributions. We choose the source term f and the boundary data g_i and h_i $(i = 1, 2)$ appropriately such that the following function satisfies the system (21),

$$u(x,y) = -\sin(\pi x^2)\sin(\pi y^2). \tag{22}$$

The distribution of this exact solution in the xy plane is illustrated by Figure 3(a).

We employ the HLConcELM method from Section 2 to solve the system (21). Let the vector $\mathbf{M} = [2, m_1, \ldots, m_{L-1}, 1]$ denote the architecture of the HLConcELM, where the two input nodes represent the coordinates x and y and the single output node represents the solution u. We employ the Gaussian activation function for all the hidden nodes, as stated at the beginning of Section 3. The output layer is linear and has no bias. The number of hidden layers and the number of hidden nodes are varied, and the specific architure will be given below when discussing the results.

We employ a uniform set of $Q = Q_1 \times Q_1$ grid points on the domain Ω, with Q_1 uniform points on each side of the boundary, as the collocation points for

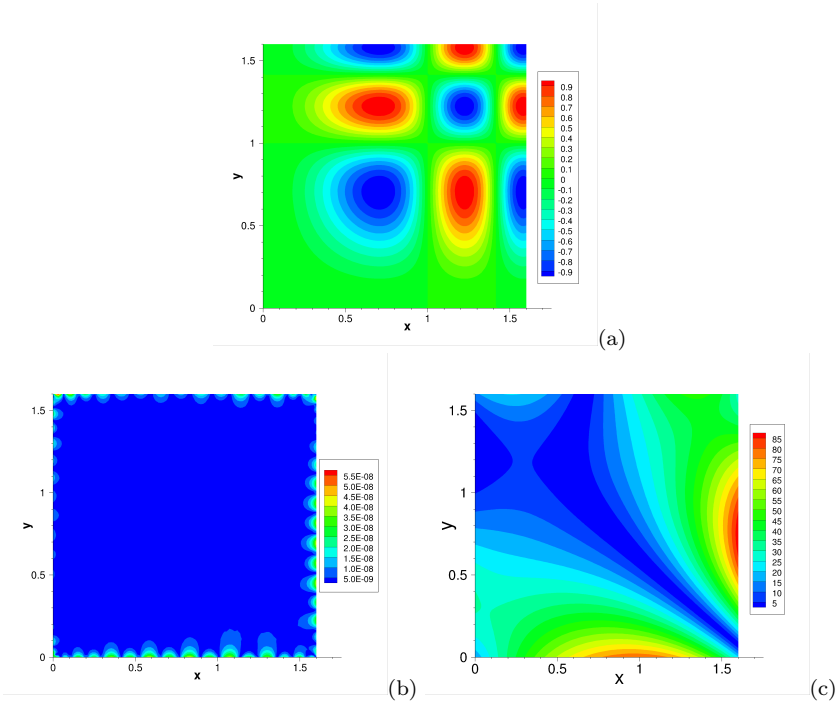

Fig. 3 Variable-coefficient Poisson equation: distributions of (a) the exact solution, (b) the absolute error of the HLConcELM solution, and (c) the absolute error of the conventional ELM solution. In (b,c), network architecture $\mathbf{M} = [2, 800, 50, 1]$, Gaussian activation function, $Q = 35 \times 35$ uniform collocation points. $\mathbf{R} = (3.0, 0.005)$ for HLConcELM in (b). $R_m = R_{m0} = 0.35$ for conventional ELM in (c), where R_{m0} is the optimal R_m computed using the method of [13].

training the neural network. Q_1 is varied in the tests. As discussed earlier, after the neural network is trained, we evaluate the neural network on another finer set of $Q_{eval} = Q_2 \times Q_2$, with $Q_2 = 101$, uniform grid points on Ω and compute the HLConcELM errors.

Figures 3(b) and (c) show a comparison of the point-wise absolute-error distributions in the xy plane of the HLConcELM solution and the conventional ELM solution obtained using a neural network with a narrow last hidden layer. Note that in conventional ELM the usual feedforward neural network has been employed (see Figure 1(a)). For both HLConcELM and conventional ELM we employ here a neural network with the architecture $\mathbf{M} = [2, 800, 50, 1]$ and a uniform set of $Q = 35 \times 35$ collocation points for the network training. With HLConcELM, for setting the random hidden-layer coefficients, we employ a hidden magnitude vector $\mathbf{R} = (3.0, 0.005)$, which is close to the optimum $\mathbf{R}^*$ obtained based on the method of [13]. With conventional ELM, we set the hidden-layer coefficients to uniform random values generated on the interval $[-R_m, R_m]$ with $R_m = R_{m0}$, where $R_{m0} = 0.35$ is the optimal R_m obtained using the method of [13] for this case. Because the last hidden layer is quite narrow (with 50 nodes), we observe that the result of the conventional ELM exhibits no accuracy, with a maximum error

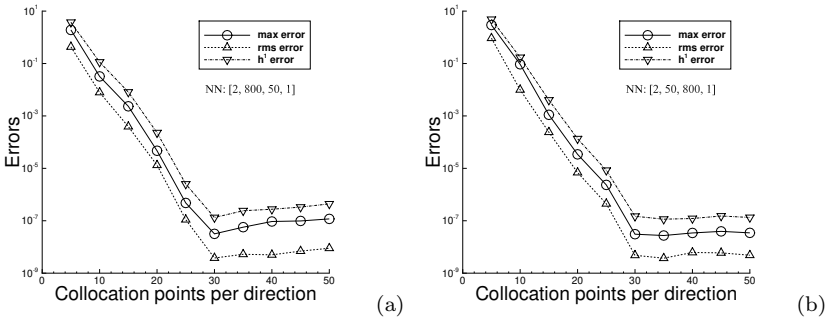

Fig. 4 Variable-coefficient Poisson equation: the maximum, rms and h^1 errors of the HLConcELM solution versus the number of collocation points per direction obtained with a network architecture of (a) $[2, 800, 50, 1]$ and (b) $[2, 50, 800, 1]$. $\mathbf{R} = (3.0, 0.005)$ in (a), and $\mathbf{R} = (0.68, 0.82)$ in (b).

around 85 in the domain. In contrast, the HLConcELM method produces highly accurate results, with the maximum error on the order of 10^{-8} in the domain.

Figure 4 illustrates the convergence behavior of the HLConcELM solution with respect to the number of collocation points in the network training. Two neural networks are considered, with the architectures given by $\mathbf{M}_1 = [2, 800, 50, 1]$ and $\mathbf{M}_2 = [2, 50, 800, 1]$, respectively. We vary the number of collocation points per direction (i.e. Q_1) systematically between $Q_1 = 5$ and $Q_1 = 50$, and record the corresponding HLConcELM errors. Figures 4(a) and (b) show the maximum, rms and h^1 errors of HLConcELM as a function of Q_1 for the two neural networks. Here the h^1 error is defined as

$$\sqrt{\frac{1}{Q_{eval}} \sum_{i=1}^{Q_{eval}} \left[(u|_{\mathbf{x}_i} - u_{ex}|_{\mathbf{x}_i})^2 + \left(\frac{\partial u}{\partial x}\bigg|_{\mathbf{x}_i} - \frac{\partial u_{ex}}{\partial x}\bigg|_{\mathbf{x}_i} \right)^2 + \left(\frac{\partial u}{\partial y}\bigg|_{\mathbf{x}_i} - \frac{\partial u_{ex}}{\partial y}\bigg|_{\mathbf{x}_i} \right)^2 \right]}$$

where u and u_{ex} denote the ELM solution and the exact solution, respectively, and $\mathbf{x}_i$ $(1 \leqslant i \leqslant Q_{eval})$ denote the evaluation points. For the network $\mathbf{M}_1$ we employ a hidden magnitude vector $\mathbf{R} = (3.0, 0.005)$, and for the network $\mathbf{M}_2$ we employ a hidden magnitude vector $\mathbf{R} = (0.68, 0.82)$. These $\mathbf{R}$ values are obtained using the method of [13]. The results indicate that the HLConcELM errors decrease approximately exponentially with increasing number of collocation points (when $Q_1 \leqslant 30$). The errors stagnate as Q_1 increases further, because of the fixed network size. Note that the last hidden layer of the network $\mathbf{M}_1$ is quite narrow (50 nodes), while that of the network $\mathbf{M}_2$ is quite wide (800 nodes). The HLConcELM method produces accurate results with both types of neural networks.

Figure 5 illustrates the convergence behavior, as well as the network training time, of the HLConcELM method with respect to the number of nodes in the neural network. We consider two groups of neural networks, with the architectures given by $\mathbf{M}_1 = [2, M, 50, 1]$ and $\mathbf{M}_2 = [2, 50, M, 1]$, respectively, where M is varied systematically between $M = 100$ and $M = 1000$. For all the test cases, we employ a fixed uniform set of $Q = 35 \times 35$ collocation points to train the neural network. For generating the hidden-layer coefficients, we use a hidden magnitude vector $\mathbf{R} = (3.0, 0.005)$ with the first group of networks $\mathbf{M}_1$, and a vector

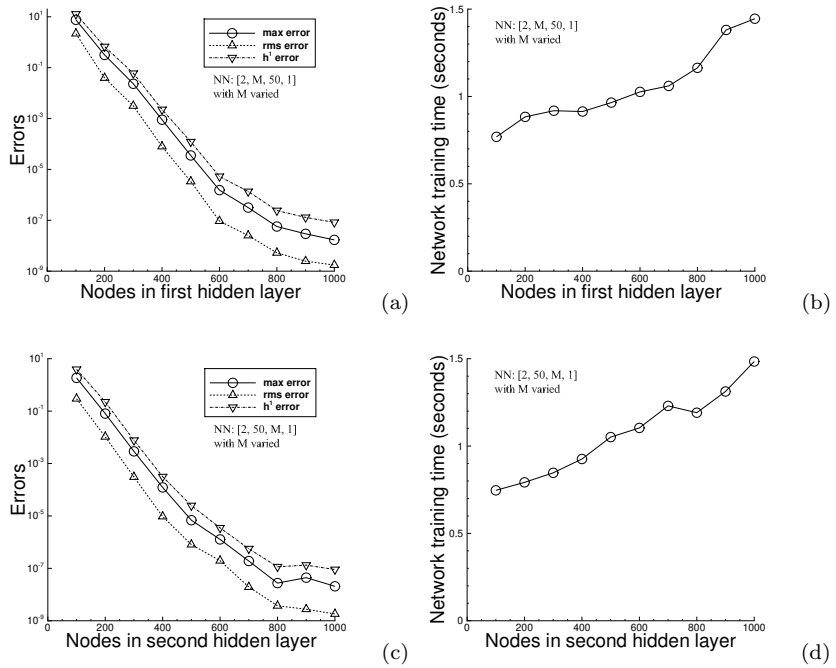

Fig. 5 Variable-coefficient Poisson equation: (a) The maximum/rms/h^1 errors and (b) the network training time of HLConcELM versus the number of nodes in the first hidden layer for a network architecture $[2, M, 50, 1]$ (varying M). (c) The maximum/rms/h^1 errors and (d) the network training time of HLConcELM versus the number of nodes in the second hidden layer for a network architecture $[2, 50, M, 1]$ (varying M). $Q = 35 \times 35$ uniform collocation points in (a,b,c,d). $\mathbf{R} = (3.0, 0.005)$ in (a,b), and $\mathbf{R} = (0.68, 0.82)$ in (c,d).

$\mathbf{R} = (0.68, 0.82)$ with the second group of networks $\mathbf{M}_2$. Figures 5(a) and (c) depict the maximum/rms/h^1 errors of HLConcELM as a function of M for these two groups of neural networks. Figures 5(b) and (d) depict the corresponding wall time it takes to train these neural networks with HLConcELM. It can be observed that the HLConcELM errors decrease approximately exponentially with increasing M (before saturation). When M becomes large the HLConcELM results are highly accurate. The network training time of the HLConcELM method increases approximately linearly with increasing M. In the range of M values tested here, it takes around a second to train the neural network to attain the HLConcELM results.

Table 1 provides a comparison of the HLConcELM accuracy and the conventional ELM accuracy for solving the variable-coefficient Poisson equation on two network architectures, $\mathbf{M}_1 = [2, 800, 50, 1]$ and $\mathbf{M}_2 = [2, 50, 800, 1]$. The network $\mathbf{M}_1$ contains a relatively small number of nodes in its last hidden layer, and the conventional ELM would not perform well. The network $\mathbf{M}_2$ contains a large number of nodes in its last hidden layer, and the conventional ELM should perform quite well. We consider a sequence of uniform collocation points, ranging from $Q = 5 \times 5$ to $Q = 30 \times 30$. Table 1 lists the maximum, rms and h^1 errors of the HLConcELM solution and the conventional ELM solution corresponding to each

network	collocation points	current HLConcELM max error	current HLConcELM rms error	current HLConcELM h^1 error	conventional ELM max error	conventional ELM rms error	conventional ELM h^1 error
[2,800,50,1]	5×5	1.91E+0	4.31E-1	3.75E+0	3.06E+1	6.34E+0	5.37E+1
	10×10	3.22E-2	7.88E-3	1.14E-1	5.48E+1	1.78E+1	8.00E+1
	15×15	2.33E-3	3.92E-4	8.15E-3	6.24E+1	2.11E+1	8.85E+1
	20×20	4.70E-5	1.32E-5	2.29E-4	6.97E+1	2.42E+1	9.62E+1
	25×25	4.78E-7	1.10E-7	2.52E-6	7.67E+1	2.70E+1	1.03E+2
	30×30	3.17E-8	3.79E-9	1.33E-7	8.31E+1	2.96E+1	1.10E+2
[2,50,800,1]	5×5	2.95E+0	9.26E-1	4.85E+0	2.48E+0	8.85E-1	5.72E+0
	10×10	9.35E-2	9.93E-3	1.71E-1	1.32E-1	1.50E-2	2.33E-1
	15×15	1.11E-3	2.40E-4	4.04E-3	6.52E-3	1.00E-3	1.55E-2
	20×20	3.42E-5	6.91E-6	1.35E-4	7.63E-5	1.33E-5	2.71E-4
	25×25	2.34E-6	4.45E-7	8.44E-6	1.83E-6	4.14E-7	8.13E-6
	30×30	3.07E-8	4.81E-9	1.48E-7	9.87E-8	2.02E-8	5.07E-7

Table 1 Variable-coefficient Poisson equation: Comparison of the maximum, rms and h^1 errors computed using the current HLConcELM method and the conventional ELM method. The HLConcELM data in this table correspond to a portion of those in Figure 4(a) for the network $[2, 800, 50, 1]$ and to those in Figure 4(b) for the network $[2, 50, 800, 1]$. For conventional ELM, the random hidden-layer coefficients are assigned to uniform random values generated on $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m obtained using the method of [13], with $R_{m0} = 0.35$ for the network $[2, 800, 50, 1]$ and $R_{m0} = 0.75$ for the network $[2, 50, 800, 1]$.

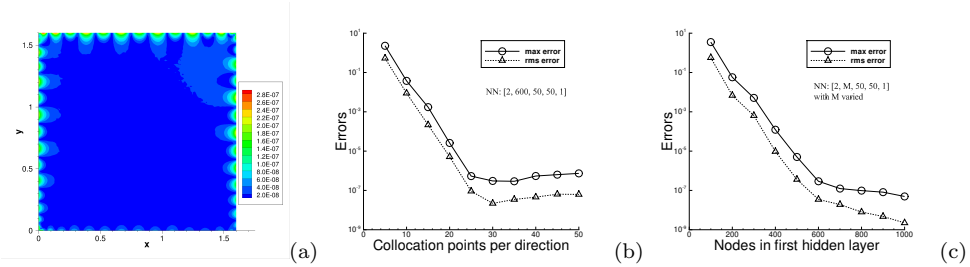

Fig. 6 Variable-coefficient Poisson equation (3 hidden layers in NN): (a) Distribution of the absolute error of the HLConcELM solution. The maximum/rms errors of HLConcELM versus (b) the number of collocation points per direction, and (c) the number of nodes in the first hidden layer of the neural network. Neural network architecture $\mathbf{M} = [2, M, 50, 50, 1]$, $Q = Q_1 \times Q_1$ uniform collocation points. $M = 600$ in (a,b) and is varied in (c). $Q_1 = 35$ in (a,c) and is varied in (b). $\mathbf{R} = (2.6, 0.005, 0.8)$ in (a,b,c).

set of collocation points. The data indicate that the conventional ELM exhibits no accuracy with the network $\mathbf{M}_1$, and exhibits exponentially increasing accuracy with increasing collocation points on the network $\mathbf{M}_2$. On the other hand, the current HLConcELM method exhibits exponentially increasing accuracy with increasing collocation points on both networks $\mathbf{M}_1$ and $\mathbf{M}_2$.

Figure 6 is an illustration of the HLConcELM results obtained on neural networks with three hidden layers. Here we consider a network architecture $\mathbf{M} = [2, M, 50, 50, 1]$, with M either fixed at $M = 600$ or varied systematically between $M = 100$ and $M = 1000$. The set of collocation points (uniform) is either fixed at $Q = 35 \times 35$ or varied systematically between $Q = 5 \times 5$ and $Q = 50 \times 50$. We employ a fixed hidden magnitude vector $\mathbf{R} = (2.6, 0.005, 0.8)$, obtained using the method of [13]. Figure 6(a) shows the HLConcELM error distribution corresponding to $\mathbf{M} = [2, 600, 50, 50, 1]$ and $Q = 35 \times 35$, indicating a quite high accuracy, with the maximum error in the domain on the order 10^{-7}. Figures 6(b) and (c) demon-

# hidden layers	**R**	Q = 15×15		Q = 30×30	
		max-err	rms-err	max-err	rms-err
1	(2.0)	8.91E+0	1.48E+0	1.48E+1	1.92E+0
2	(0.73,0.023)	4.86E-2	4.69E-3	1.10E-1	1.26E-2
3	(0.77,0.07,0.17)	2.57E-3	8.09E-4	1.34E-3	1.10E-4
4	(0.6,0.19,0.16,0.05)	1.67E-3	4.07E-4	6.69E-5	7.20E-6
5	(0.48,0.23,0.25,0.14,0.3)	1.84E-3	3.81E-4	5.54E-6	7.33E-7
6	(0.54,0.24,0.18,0.13,0.38,0.39)	7.66E-4	1.54E-4	9.61E-7	8.60E-8
7	(0.58,0.25,0.12,0.25,0.57,1.18,0.12)	2.74E-3	6.77E-4	9.19E-7	5.46E-8

Table 2 Variable-coefficient Poisson equation: the maximum and rms errors of HLConcELM versus the number of hidden layers (i.e. depth) in the neural network architecture, with two uniform sets of collocation points ($Q = 15 \times 15$ and 30×30). The width of each hidden layer is 100. For example, the network architecture is $[2, 100, 100, 100, 100, 1]$ with 4 hidden layers. The hidden magnitude vector **R** is listed in the table.

strate the exponential convergence (before saturation) of the HLConcELM errors with respect to the collocation points Q_1 and the number of nodes M, respectively. These results show that the current HLConcELM method can produce highly accurate results on neural networks with multiple hidden layers and a narrow last hidden layer.

Table 2 illustrates the effect of the neural-network depth (number of hidden layers) on the HLConcELM accuracy. Here we vary the number of hidden layers systematically between 1 and 7, while the number of nodes in each hidden layer is fixed at 100. The hidden magnitude vector **R** is provided in the table for each network architecture. The maximum and rms errors of the HLConcELM solutions for two sets of collocation points are listed in the table. For each set of collocation points, the HLConcELM errors decrease approximately exponentially initially with increasing number of hidden layers, and then stagnate when the depth increases beyond a certain level. With $Q = 15 \times 15$ collocation points, the HLConcELM maximum error reaches a level 10^{-3} with 3 or more hidden layers. With $Q = 30 \times 30$, the maximum error reaches a level 10^{-7} with 6 or more hidden layers.

3.2 Advection Equation

In the next example we employ the 1D advection equation (plus time) to test the HLConcELM method. Consider the spatial-temporal domain, $(x, t) \in \Omega = [0, 5] \times [0, 40]$, and the following initial/boundary value problem on Ω,

$$\frac{\partial u}{\partial t} - 2\frac{\partial u}{\partial x} = 0, \tag{23a}$$

$$u(0, t) = u(5, t), \tag{23b}$$

$$u(x, 0) = 20 \tanh\left(\frac{1}{10}\cos\left(\frac{2\pi}{5}(x - 3)\right)\right). \tag{23c}$$

In the above equations $u(x, t)$ is the field function to be solved for, and we impose the periodic boundary condition in the spatial direction. This system has the following exact solution,

$$u(x, t) = 20 \tanh\left(\frac{1}{10}\cos\left(\frac{2\pi}{5}(x + 2t - 3)\right)\right). \tag{24}$$

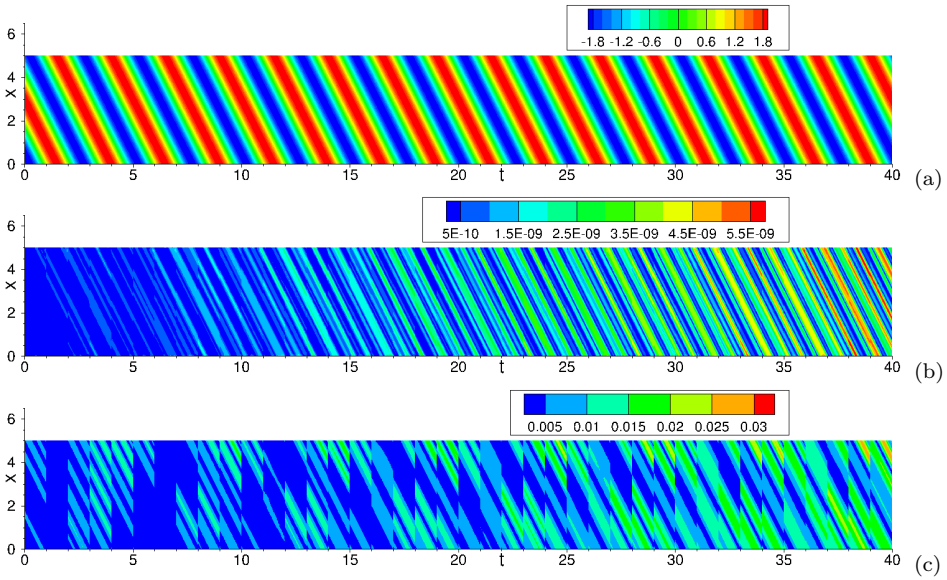

Fig. 7 Advection equation: Distributions in the spatial-temporal domain of (a) the exact solution, (b) the absolute error of the HLConcELM solution, and (c) the absolute error of the conventional ELM solution. In (b,c), network architecture [2, 500, 50, 1], 40 uniform time blocks, $Q = 35 \times 35$ uniform collocation points per time block. $\mathbf{R} = (3.0, 1.0)$ in (b) for HLConcELM. $R_m = R_{m0} = 0.065$ in (c) for conventional ELM.

The distribution of this solution on the spatial-temporal domain is illustrated in Figure 7(a).

To solve the system (23), we employ the HLConcELM method combined with the block time marching scheme (see Remark 5 and [9]). We divide the domain Ω into 40 uniform time blocks in time. For computing each time block with HLConcELM, we employ a network architecture $\mathbf{M} = [2, m_1, \ldots, m_{L-1}, 1]$, where the two input nodes represent x and t and the single output node represents $u(x, t)$. Let $Q = Q_1 \times Q_1$ denote the uniform set of collocation points for each time block (Q_1 grid points in both x and t directions), where Q_1 is varied in the tests. As discussed before, upon completion of training, the neural network is evaluated on a uniform set of $Q_{eval} = 101 \times 101$ grid points on each time block and the corresponding errors are computed. The maximum and rms errors reported below refer to the errors of the HLConcELM solution on the entire domain Ω (over 40 time blocks).

Figures 7(b) and (c) illustrate the absolute-error distributions on Ω of the HLConcELM solution and the conventional ELM solution, respectively. For both methods, we employ 40 time blocks in block time marching, a neural network architecture $\mathbf{M} = [2, 500, 50, 1]$ with the Gaussian activation function, and a set of $Q = 35 \times 35$ uniform collocation points per time block. For HLConcELM we employ $\mathbf{R} = (3.0, 1.0)$, which is computed by the method of [13]. For conventional ELM we employ $R_m = R_{m0} = 0.065$, which is also obtained by the method of [13], for generating the random hidden-layer coefficients. Because the number of nodes in the last hidden layer is quite small, the conventional ELM exhibits a low accuracy, with the maximum error on the order of 10^{-2} in the domain. On the other hand,

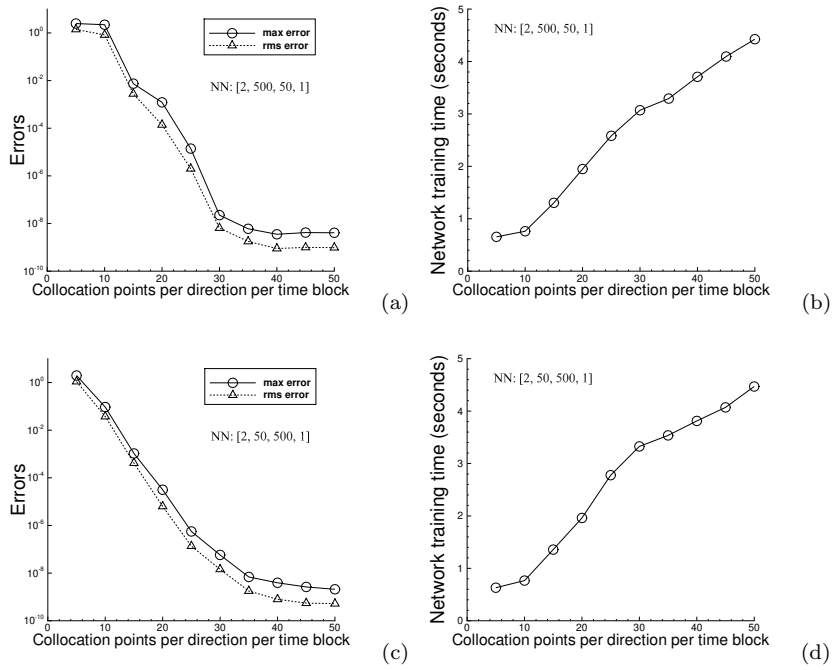

Fig. 8 Advection equation: The maximum/rms errors in the domain Ω (a,c) and the network training time (b,d) of HLConcELM versus the number of collocation points per direction in each time block. The results are attained with two network architectures: (a,b) $\mathbf{M}_1 = [2, 500, 50, 1]$, and (c,d) $\mathbf{M}_2 = [2, 50, 500, 1]$. $\mathbf{R} = (3.0, 1.0)$ in (a,b) for the network $\mathbf{M}_1$, and $\mathbf{R} = (0.9, 0.5)$ in (c,d) for the network $\mathbf{M}_2$.

the HLConcELM method produces a highly accurate solution, with the maximum error on the order of 10^{-9} in the domain.

Figure 8 illustrates the convergence behavior, as well as the growth in the network training time, of the HLConcELm method with respect to the number of collocation points. We have considered two network architectures, $\mathbf{M}_1 = [2, 500, 50, 1]$ and $\mathbf{M}_2 = [2, 50, 500, 1]$, with a narrower last hidden layer in $\mathbf{M}_1$ and a wider one in $\mathbf{M}_2$. A uniform set of $Q = Q_1 \times Q_1$ collocation points is employed, with Q_1 varied systematically between $Q_1 = 5$ and $Q_1 = 50$ in the tests. The hidden magnitude vector $\mathbf{R}$ computed by the method [13] is used in the simulations, with $\mathbf{R} = (3.0, 1.0)$ for the network $\mathbf{M}_1$ and $\mathbf{R} = (0.9, 0.5)$ for the network $\mathbf{M}_2$. Figures 8(a) and (b) depict the maximum/rms errors on Ω and the network training time, respectively, as a function of Q_1 obtained with the neural network $\mathbf{M}_1$. Figures 8(c) and (d) show the corresponding results obtained with the network $\mathbf{M}_2$. While the convergence behavior is not quite regular, one can observe that the HLConcELM errors approximately decrease exponentially (before saturation) with increasing number of collocation points. The network training time grows approximately linearly with increasing number of training collocation points.

Figure 9 illustrates the convergence behavior of the HLConcELM method with respect to the number of nodes in the neural network. Two groups of neural net-

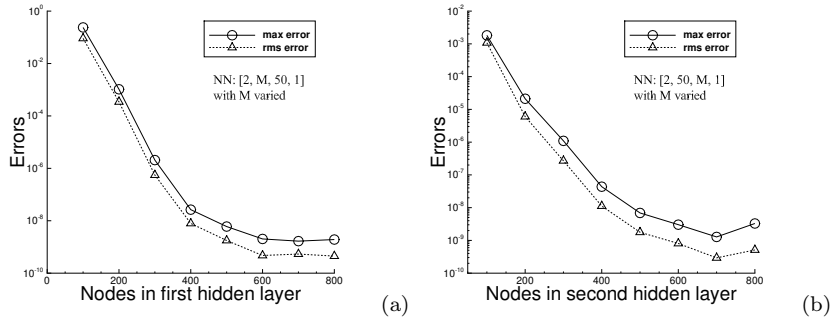

Fig. 9 Advection equation: (a) The HLConcELM maximum/rms errors on Ω versus the number of nodes in the first hidden layer for the network architecture $\mathbf{M}_1 = [2, M, 50, 1]$ (varying M). (b) The HLConcELM maximum/rms errors on Ω versus the number of nodes in the second hidden layer for the network architecture $\mathbf{M}_2 = [2, 50, M, 1]$ (varying M). $Q = 35 \times 35$ in (a,b). $\mathbf{R} = (3.0, 1.0)$ in (a) for the network $\mathbf{M}_1$, and $\mathbf{R} = (0.9, 0.5)$ in (b) for the network $\mathbf{M}_2$.

network architecture	collocation points	current max error	current rms error	HLConcELM max error	HLConcELM rms error	conventional max error	conventional rms error	ELM max error	ELM rms error
$[2, 500, 50, 1]$	5×5	$2.48E+0$	$1.41E+0$	$1.33E+0$	$4.57E-1$				
	10×10	$2.21E+0$	$8.26E-1$	$5.97E-2$	$1.91E-2$				
	15×15	$7.45E-3$	$2.78E-3$	$4.31E-2$	$1.30E-2$				
	20×20	$1.22E-3$	$1.39E-4$	$3.62E-2$	$1.08E-2$				
	25×25	$1.39E-5$	$2.00E-6$	$3.15E-2$	$9.50E-3$				
	30×30	$2.25E-8$	$6.47E-9$	$3.24E-2$	$8.89E-3$				
$[2, 50, 500, 1]$	5×5	$1.97E+0$	$1.10E+0$	$1.86E+0$	$9.18E-1$				
	10×10	$9.33E-2$	$3.74E-2$	$4.18E-2$	$1.65E-2$				
	15×15	$1.05E-3$	$4.16E-4$	$3.09E-4$	$8.48E-5$				
	20×20	$3.16E-5$	$6.30E-6$	$1.97E-4$	$5.01E-5$				
	25×25	$5.60E-7$	$1.36E-7$	$7.60E-5$	$5.85E-6$				
	30×30	$5.80E-8$	$1.45E-8$	$9.76E-8$	$3.32E-8$				

Table 3 Advection equation: Comparison of the maximum/rms errors on Ω from the current HLConcELM method and the conventional ELM method [9]. The HLConcELM data in this table correspond to a portion of those in Figure 8(a) for the network $[2, 500, 50, 1]$ and to those in Figure 8(c) for the network $[2, 50, 500, 1]$. For conventional ELM, the hidden-layer coefficients are set to uniform random values generated on $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m computed by the method of [13], with $R_{m0} = 0.065$ for the network $[2, 500, 50, 1]$ and $R_{m0} = 0.65$ for the network $[2, 50, 500, 1]$.

works are considered in these tests, with an architecture $\mathbf{M}_1 = [2, M, 50, 1]$ for the first group and $\mathbf{M}_2 = [2, 50, M, 1]$ for the second one, with M varied systematically. A uniform set of $Q = 35 \times 35$ collocation points is employed for training the neural networks. We use $\mathbf{R} = (3.0, 1.0)$ for the architecture $\mathbf{M}_1$ and $\mathbf{R} = (0.9, 0.5)$ for the architecture $\mathbf{M}_2$. The plots (a) and (b) show the maximum/rms errors of HLConcELM on Ω as a function of M, indicating that the errors decrease approximately exponentially (before saturation) with increasing M in the neural network.

Table 3 provides an accuracy comparison of the current HLConcELM method and the conventional ELM method [9] for solving the advection equation. Two neural networks are considered here, with the architectures given by $\mathbf{M}_1 = [2, 500, 50, 1]$ and $\mathbf{M}_2 = [2, 50, 500, 1]$, respectively. The maximum/rms errors of both methods

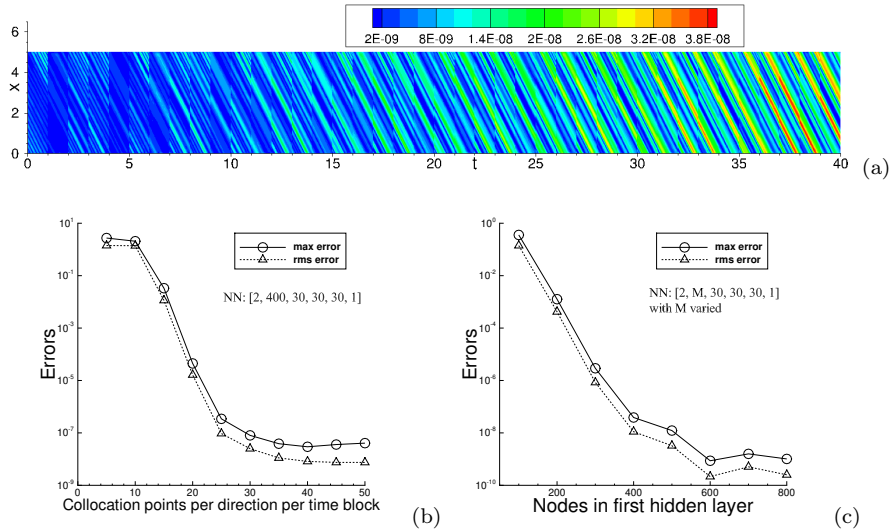

Fig. 10 Advection equation (4 hidden layers in neural network): (a) Error distribution of the HLConcELM solution on Ω. The HLConcELM maximum/rms errors on Ω versus (b) the number of collocation points per direction in each time block, and (c) the number of nodes in the first hidden layer (M). Network architecture $[2, M, 30, 30, 30, 1]$, 40 time blocks in block time marching. $Q = 35 \times 35$ in (a,c), and is varied in (b). $M = 400$ in (a,b), and is varied in (c). $\mathbf{R} = (3.1, 1.0, 0.9, 0.8)$ in (a,b,c).

on the domain Ω corresponding to a sequence of collocation points are listed in the table. With the network $\mathbf{M}_1$, whose last hidden layer is narrower, the conventional ELM exhibits only a fair accuracy with increasing collocation points, with its maximum errors on the order of 10^{-2}. In contrast, the current HLConcELM method produces highly accurate results with the network $\mathbf{M}_1$, with the maximum error reaching the order of 10^{-8} on the larger set of collocation points. With the network $\mathbf{M}_2$, whose last hidden layer is wider, both the conventional ELM and the current HLConcELM produce highly accurate results with increasing number of collocation points. These observations are consistent with those in the previous subsection for the variable-coefficient Poisson equation.

Figure 10 illustrates the HLConcELM results obtained on a deeper neural network containing 4 hidden layers for solving the advection equation. The network architecture is given by $\mathbf{M} = [2, M, 30, 30, 30, 1]$, where M is either fixed at $M = 400$ or varied systematically between $M = 100$ and $M = 800$. A uniform set of $Q = Q_1 \times Q_1$ collocation points is used to train the network, where Q_1 is either fixed at $Q_1 = 35$ or varied systematically between $Q_1 = 5$ and $Q_1 = 50$. In all simulations we employ a hidden magnitude vector $\mathbf{R} = (3.1, 1.0, 0.9, 0.8)$, which is computed using the method of [13]. Figure 10(a) depicts the distribution of the absolute error of the HLConcELM solution on Ω, which corresponds to $M = 400$ and $Q_1 = 35$. It can be observed that the result is highly accurate, with a maximum error on the order of 10^{-8} in the domain. Figure 10(b) shows the maximum/rms errors of HLConcELM as a function of Q_1, with a fixed $M = 400$ in the tests. Figure 10(c) shows the maximum/rms errors of HLConcELM as a function of M in the neural network, with a fixed $Q_1 = 35$ for the collocation

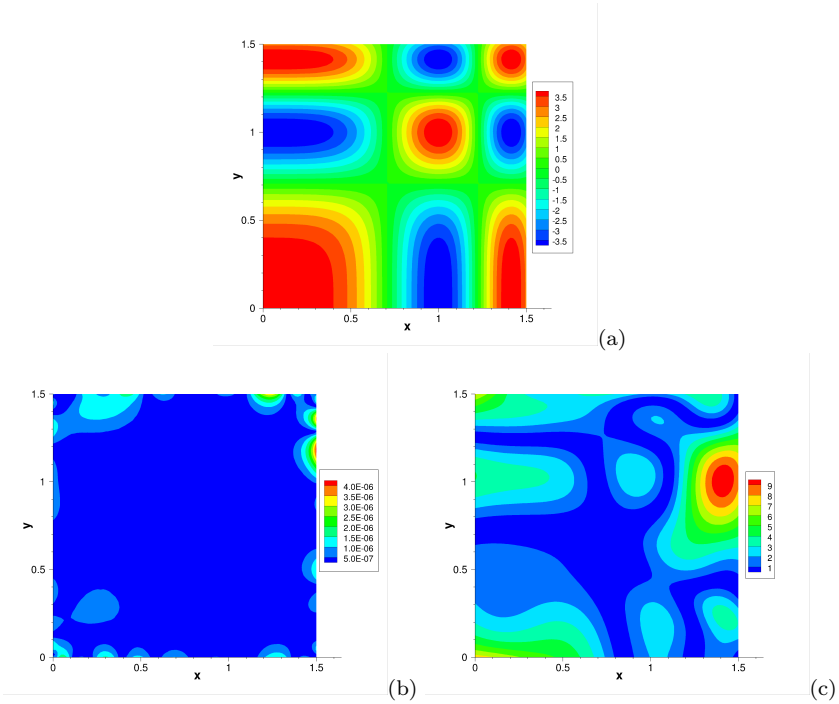

Fig. 11 Nonlinear Helmholtz equation: Distributions of (a) the exact solution, (b) the absolute error of the HLConcELM solution, and (c) the absolute error of the conventional ELM solution. In (b,c), network architecture $[2, 500, 30, 1]$, Gaussian activation function, $Q = 35 \times 35$ uniform collocation points. $\mathbf{R} = (2.0, 3.0)$ in (b) for HLConcELM. $R_m = R_{m0} = 0.6$ in (c) for conventional ELM.

points. The exponential convergence of the HLConcELM errors (before saturation) is unmistakable.

3.3 Nonlinear Helmholtz Equation

We employ a nonlinear Helmholtz equation to test the HLConcELM method for the next problem. Consider the 2D domain $(x, y) \in \Omega = [0, 1.5] \times [0, 1.5]$ and the following boundary value problem on Ω,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - 100u + 10\cosh(u) = f(x, y), \tag{25a}$$

$$u(0, y) = g_1(y), \quad u(1.5, y) = g_2(y), \quad u(x, 0) = h_1(x), \quad u(x, 1.5) = h_2(x). \tag{25b}$$

In the above equations $u(x, y)$ is the field solution to be sought, $f(x, y)$ is a prescribed source term, g_i and h_i $(i = 1, 2)$ are the Dirichlet boundary data. In this subsection we choose f, g_i and h_i $(i = 1, 2)$ such that the system (25) has the following solution,

$$u(x, y) = 4\cos\left(\pi x^2\right)\cos\left(\pi y^2\right). \tag{26}$$

The distribution of this solution in the xy plane is illustrated in Figure 11(a).

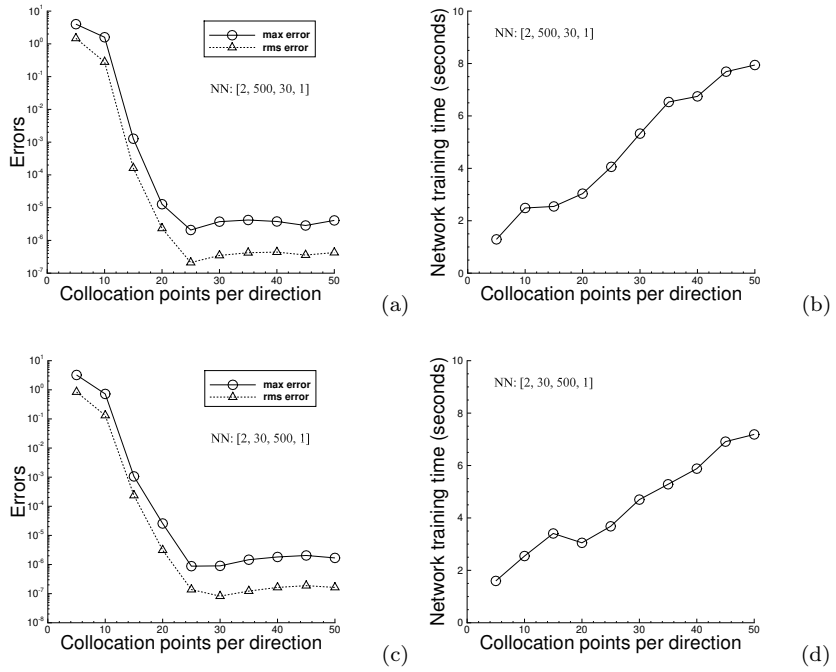

Fig. 12 Nonlinear Helmholtz equation: The maximum/rms errors (a,c) and the network training time (b,d) of the HLConcELM method versus the number of collocation points in each direction. In (a,b), network architecture $[2, 500, 30, 1]$, $\mathbf{R} = (2.0, 3.0)$. In (c,d), network architecture $[2, 30, 500, 1]$, $\mathbf{R} = (0.65, 0.7)$. In (a,b,c,d), uniform collocation points $Q = Q_1 \times Q_1$, with Q_1 varied.

We employ the HLConcELM method with neural networks that contain two input nodes, representing the x and y, and a single output node, representing the solution u. The number of hidden layers and the number of hidden nodes are varied and will be specified below. To train the neural network, we employ a uniform set of $Q = Q_1 \times Q_1$ collocation points on Ω, with Q_1 varied in the tests. The ELM errors reported below are computed on a finer set of $Q_{eval} = 101 \times 101$ uniform grid points, as explained before.

Figures 11(b) and (c) illustrate the absolute-error distributions obtained using the HLConcELM method and the conventional ELM method with the network architecture $\mathbf{M} = [2, 500, 30, 1]$. A uniform set of $Q = 35 \times 35$ collocation points has been used to train the network with both methods. The hidden magnitude vector is $\mathbf{R} = (2.0, 3.0)$ for HLConcELM, which is obtained with the method of [13]. For conventional ELM we have employed $R_m = R_{m0} = 0.6$, which is also obtained using the method of [13], for generating the random hidden-layer coefficients. The conventional ELM solution is inaccurate, with the maximum error on order of 10. On the other hand, the current HLConcELM method produces an accurate solution on the same network architecture, with the maximum error on the order of 10^{-6} in the domain.

Figure 12 illustrates the convergence behavior and the network training time with respect to the training collocation points of the HLConcELM method for

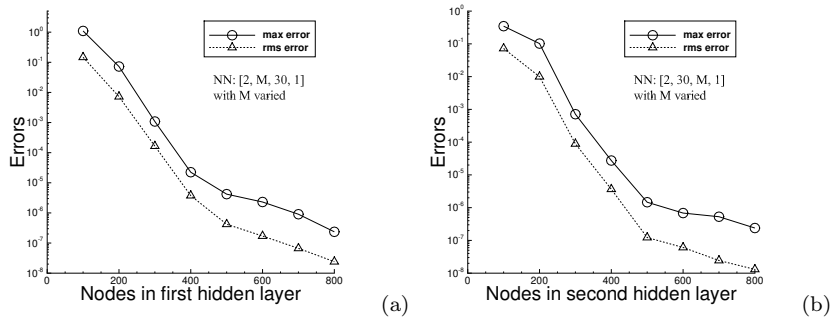

Fig. 13 Nonlinear Helmholtz equation: (a) The HLConcELM maximum/rms errors versus the number of nodes in the first hidden layer with network architecture $[2, M, 30, 1]$ (M varied). (b) The HLConcELM maximum/rms errors versus the number of nodes in the second hidden layer with the architecture $[2, 30, M, 1]$ (M varied). $Q = 35 \times 35$ collocation points in (a,b). $\mathbf{R} = (2.0, 3.0)$ in (a), and $\mathbf{R} = (0.65, 0.7)$ in (b).

solving the nonlinear Helmholtz equation. Two network architectures are considered here, $\mathbf{M}_1 = [2, 500, 30, 1]$ and $\mathbf{M}_2 = [2, 30, 500, 1]$. The number of collocation points in each direction (Q_1) is varied systematically between $Q_1 = 5$ and $Q_1 = 50$ in these tests. We employ $\mathbf{R} = (2.0, 3.0)$ for the network $\mathbf{M}_1$ and $\mathbf{R} = (0.65, 0.7)$ for the network $\mathbf{M}_2$, which are obtained using the method of [13]. Figures 12(a) and (b) show the maximum/rms errors and the network training time of the HLConcELM method as a function of Q_1 for the neural network $\mathbf{M}_1$. Figures 12(c) and (d) show the corresponding results for the network $\mathbf{M}_2$. The exponential convergence (before saturation) and the near linear growth in the network training time observed here for the nonlinear Helmholtz equation are consistent with those for the linear problems in previous subsections.

Figure 13 illustrates the error convergence of the HLConcELM method with respect to the number of nodes in the neural network. Two groups of neural networks are considered here, with the architectures $\mathbf{M}_1 = [2, M, 30, 1]$ and $\mathbf{M}_2 = [2, 30, M, 1]$, where M is varied systematically. The networks are trained on a uniform set of $Q = 35 \times 35$ collocation points. The plots (a) and (b) show the maximum/rms errors of HLConcELM as a function of M for these two groups of neural networks. It can be observed that the errors decrease approximately exponentially with increasing M.

Table 4 compares the numerical errors of the current HLConcELM method and the conventional ELM method for solving the nonlinear Helmholtz equation on two network architectures, $\mathbf{M}_1 = [2, 500, 30, 1]$ and $\mathbf{M}_2 = [2, 30, 500, 1]$, trained on a sequence of uniform sets of collocation points. The HLConcELM method produces highly accurate results on both neural networks. On the other hand, while the conventional ELM produces accurate results on the network $\mathbf{M}_2$, its solution on the network $\mathbf{M}_1$ is utterly inaccurate.

Figure 14 illustrates the HLConcELM results computed on a deeper neural network with 5 hidden layers. The network architecture is given by $\mathbf{M} = [2, M, 30, 30, 30, 30, 1]$, where M is either fixed at $M = 500$ or varied systematically in the tests. The network is trained on a uniform set of $Q = Q_1 \times Q_1$ collocation points, where Q_1 is either fixed at $Q_1 = 35$ or varied systematically. Figure 14(a)

network architecture	collocation points	current max error	HLConcELM rms error	conventional max error	ELM rms error
$[2, 500, 30, 1]$	5×5	$4.00E+0$	$1.48E+0$	$7.64E+0$	$2.41E+0$
	10×10	$1.59E+0$	$2.80E-1$	$9.69E+0$	$2.73E+0$
	15×15	$1.27E-3$	$1.62E-4$	$9.73E+0$	$2.71E+0$
	20×20	$1.27E-5$	$2.34E-6$	$9.74E+0$	$2.73E+0$
	25×25	$2.08E-6$	$2.11E-7$	$9.74E+0$	$2.73E+0$
	30×30	$3.74E-6$	$3.48E-7$	$9.75E+0$	$2.74E+0$
$[2, 30, 500, 1]$	5×5	$3.23E+0$	$8.43E-1$	$3.80E+0$	$1.15E+0$
	10×10	$7.22E-1$	$1.32E-1$	$3.08E+0$	$7.48E-1$
	15×15	$1.06E-3$	$2.36E-4$	$3.86E-4$	$6.29E-5$
	20×20	$2.56E-5$	$3.12E-6$	$3.15E-5$	$5.67E-6$
	25×25	$8.78E-7$	$1.38E-7$	$1.33E-6$	$2.68E-7$
	30×30	$8.99E-7$	$8.20E-8$	$1.76E-6$	$1.92E-7$

Table 4 Nonlinear Helmholtz equation: Comparison of the maximum/rms errors from the HLConcELM method and the conventional ELM method [9]. The HLConcELM data in this table correspond to a portion of those in Figure 12(a) for the network $[2, 500, 30, 1]$ and to those in Figure 12(c) for the network $[2, 30, 500, 1]$. For conventional ELM, the hidden-layer coefficients are set to uniform random values generated on $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m obtained using the method of [13], with $R_{m0} = 0.6$ for the network $[2, 500, 30, 1]$ and $R_{m0} = 0.65$ for the network $[2, 30, 500, 1]$.

depicts the absolute-error distribution of the HLConcELM solution obtained with $M = 500$ and $Q_1 = 35$, indicating a quite high accuracy with the maximum error on the order of 10^{-6} in the domain. Figures 14(b) and (c) show the HLConcELM errors as a function of Q_1 and M, respectively. The exponential convergence (prior to saturation) of these errors is evident.

3.4 Burgers' Equation

In the next benchmark example we use the viscous Burgers' equation to test the performance of the HLConcELM method. Consider the spatial-temporal domain, $(x, t) \in \Omega = [-1, 1] \times [0, 1]$, and the following initial/boundary value problem on Ω,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \tag{27a}$$

$$u(-1, t) = u(1, t) = 0, \tag{27b}$$

$$u(x, 0) = -\sin(\pi x), \tag{27c}$$

where $\nu = \frac{1}{100\pi}$, and $u(x, t)$ denotes the field function to be solved for. This problem has the following exact solution [2],

$$u(x, t) = -\frac{\int_{-\infty}^{\infty} \sin \pi(x - \eta) f(x - \eta) e^{-\frac{\eta^2}{4\nu t}} d\eta}{\int_{-\infty}^{\infty} f(x - \eta) e^{-\frac{\eta^2}{4\nu t}} d\eta}, \tag{28}$$

where $f(y) = e^{-\frac{\cos(\pi y)}{2\pi\nu}}$. Figure 15 illustrates the distribution of this solution on the spatial-temporal domain, which indicates that a sharp gradient develops in the domain over time.

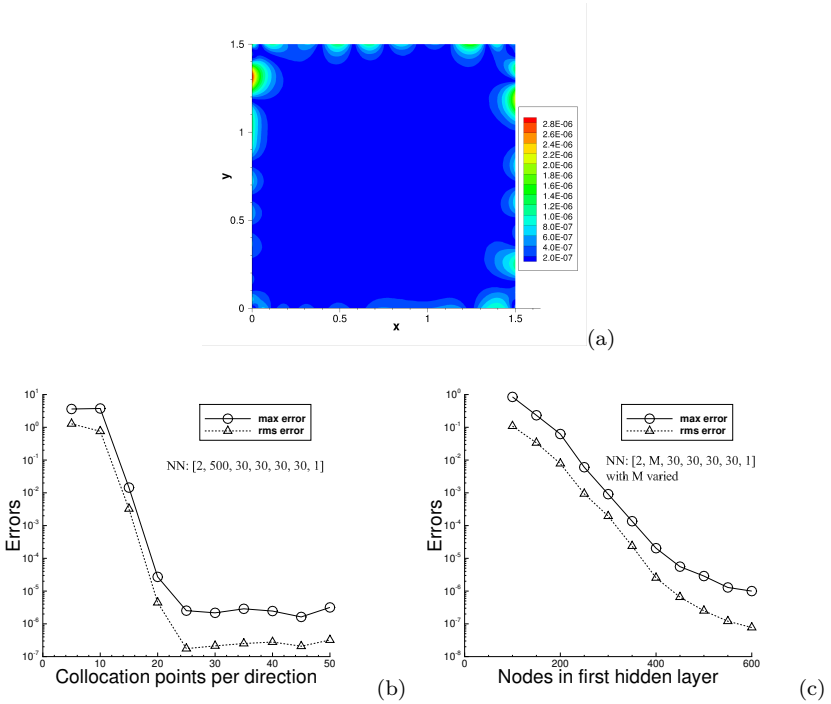

Fig. 14 Nonlinear Helmholtz equation (5 hidden layers in neural network): (a) Error distribution of the HLConcELM solution. The maximum/rms errors of the HLConcELM solution versus (b) the number of collocation points in each direction and (c) the number of nodes in the first hidden layer (M). Neural network architecture $[2, M, 30, 30, 30, 30, 1]$, $Q = Q_1 \times Q_1$ uniform collocation points. $M = 500$ in (a,b), and is varied in (c). $Q_1 = 35$ in (a,c), and is varied in (b). $\mathbf{R} = (2.1, 0.1, 2.0, 2.5, 0.5)$ in (a,b,c).

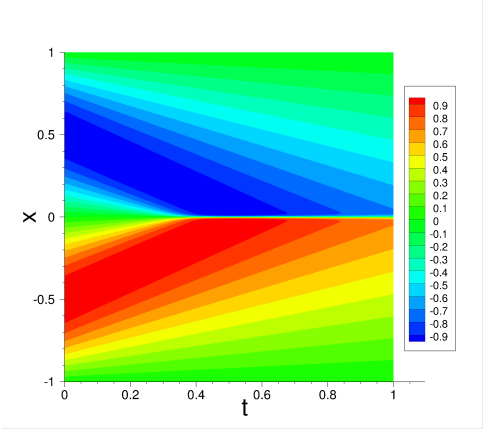

Fig. 15 Burgers' equation: distribution of the exact solution.

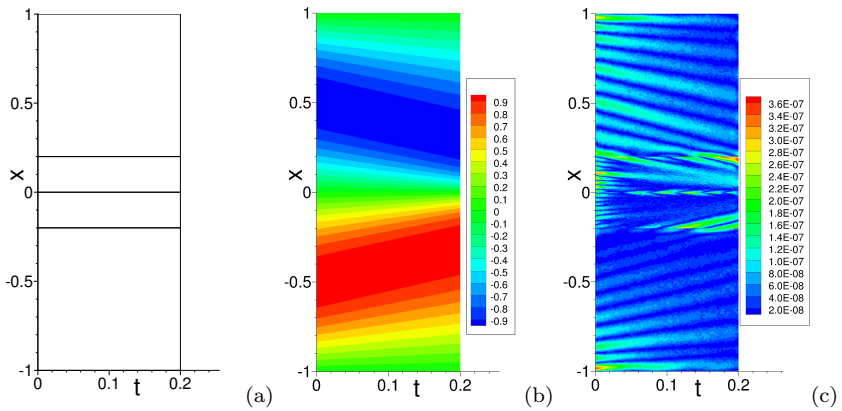

Fig. 16 Burgers's equation on the smaller domain Ω_1 ($t \in [0, 0.2]$): (a) Configuration of the 4 sub-domains in the locHLConcELM simulation. Distributions of the locHLConcELM solution (b) and its absolute error (c) on Ω_1. Local neural-network architecture: [2, 200, 30, 1], $Q = 21 \times 21$ uniform collocation points per sub-domain, $\mathbf{R} = (0.9, 0.05)$.

We will first solve the problem (27) on a smaller domain (with a smaller temporal dimension) $\Omega_1 = [-1, 1] \times [0, 0.2]$, before the sharp gradient develops, in order to investigate the convergence behavior of the HLConcELM method. Then we will compute this problem on the larger domain Ω using HLConcELM.

On the smaller domain Ω_1 we solve the system (27) by the locHLConcELM method (local version of HLConcELM, see Remark 6). We partition Ω_1 along the x direction into 4 sub-domains; see Figure 16(a). These sub-domains are non-uniform, and the x coordinates of the sub-domain boundaries are given by the vector $\boldsymbol{\mathcal{X}} = [-1, -0.2, 0, 0.2, 1]$. We impose C^1 continuity conditions in x across the interior sub-domain boundaries. We employ a HLConcELM for the local neural network on each sub-domain, which contains two input nodes (representing the x and t of the sub-domain) and a single output node (representing the solution u on the sub-domain). The specific architectures of the neural network will be provided below. On each sub-domain we employ a uniform set of $Q = Q_1 \times Q_1$ collocation points (Q_1 points in both x and t directions) for the network training, with Q_1 varied in the tests. We train the overall neural network, which consists of the local neural networks coupled together by the C^1 continuity conditions, by the nonlinear least squares method; see Section 2.3 and also [9].

Figures 16(b) and (c) illustrate the distributions of the HLConcELM solution and its absolute error on the domain Ω_1, respectively. These results are obtained by locHLConcELMs with an architecture $[2, 200, 30, 1]$ and a uniform set of $Q = 21 \times 21$ collocation points on each sub-domain. The hidden magnitude vector is $\mathbf{R} = (0.9, 0.05)$, which is obtained using the method of [13]. The locHLConcELM method produces an accurate solution, with the maximum error on the order of 10^{-7} on Ω_1.

Figure 17 illustrates the convergence behavior and the network training time of the locHLConcELM method with respect to the increase of the collocation points for the smaller domain Ω_1. The local network architecture is given by $[2, 200, 30, 1]$, and the collocation points are varied systematically between $Q = 5 \times 5$ and $Q = 30 \times 30$ in the tests. The plots (a) and (b) show the locHLConcELM er-

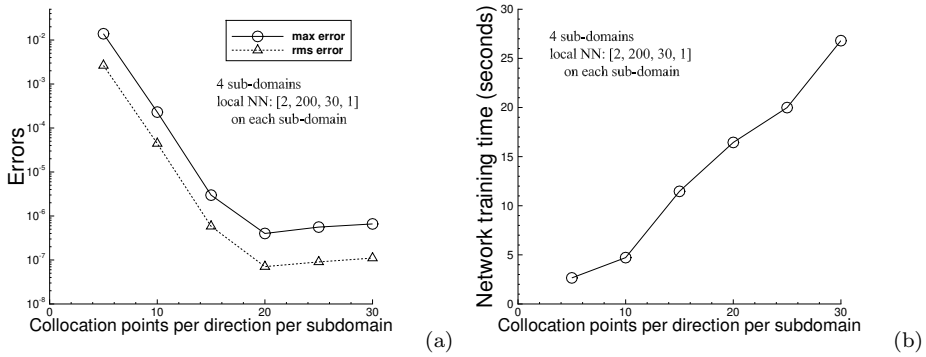

Fig. 17 Burgers' equation on the smaller domain Ω_1: (a) The locHLConcELM maximum/rms errors and (b) the network training time versus the number of collocation points per direction in each sub-domain. Local network architecture: [2, 200, 30, 1], $Q = Q_1 \times Q_1$ uniform collocation points (Q_1 varied), $\mathbf{R} = (0.9, 0.05)$.

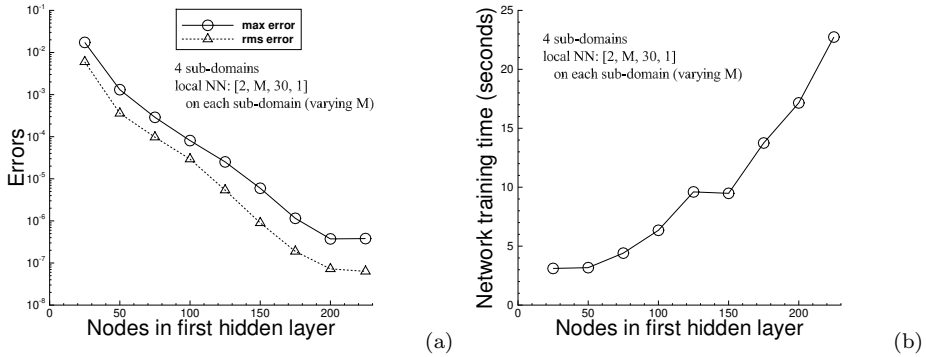

Fig. 18 Burgers' equation on the smaller domain Ω_1: (a) The locHLConcELM maximum/rms errors and (b) the network training time versus the number of nodes in the first hidden layer (M). Local network architecture [2, M, 30, 1] (M varied), $Q = 21 \times 21$ uniform collocation points per sub-domain, $\mathbf{R} = (0.9, 0.05)$.

rors and the network training time as a function of the number of collocation points in each direction, respectively. We observe that the locHLConcELM errors decrease exponentially (before saturation) and the network training time grows approximately linearly with increasing collocation points.

Figure 18 is an illustration of the convergence behavior and the network training time of the locHLConcELM method with respect to the size of the neural network. The local network architecture is given by $[2, M, 30, 1]$, with M varied systematically. We employ a fixed uniform set of $Q = 21 \times 21$ collocation points, and a hidden magnitude vector $\mathbf{R} = (0.9, 0.05)$ obtained using the method of [13]. One observes that the errors decrease exponentially and that the network training time grows superlinearly with increasing M.

Table 5 provides an accuracy comparison of the HLConcELM method and the conventional locELM method [9] for solving the Burgers' equation on the

local network architecture	collocation points	current max error	locHLConcELM rms error	conventional max error	locELM rms error
$[2, 200, 30, 1]$	5×5	$1.39E-2$	$2.64E-3$	$4.12E-2$	$1.21E-2$
	10×10	$2.30E-4$	$4.44E-5$	$6.62E-2$	$2.84E-2$
	15×15	$2.98E-6$	$5.83E-7$	$7.42E-2$	$3.16E-2$
	20×20	$4.01E-7$	$7.06E-8$	$7.97E-2$	$3.39E-2$
	25×25	$5.59E-7$	$9.04E-8$	$8.44E-2$	$3.58E-2$
	30×30	$6.60E-7$	$1.11E-7$	$8.86E-2$	$3.76E-2$

Table 5 Burgers' equation on the smaller domain Ω_1: Comparison of the maximum/rms errors from the locHLConcELM method and the conventional locELM method [9]. The locHLConcELM data in this table correspond to those in Figure 17(a). For conventional locELM, the random hidden-layer coefficients are set to uniform random values generated on $[-R_m, R_m]$ with $R_m = R_{m0} = 0.175$, where R_{m0} is the optimal R_m obtained using the method of [13].

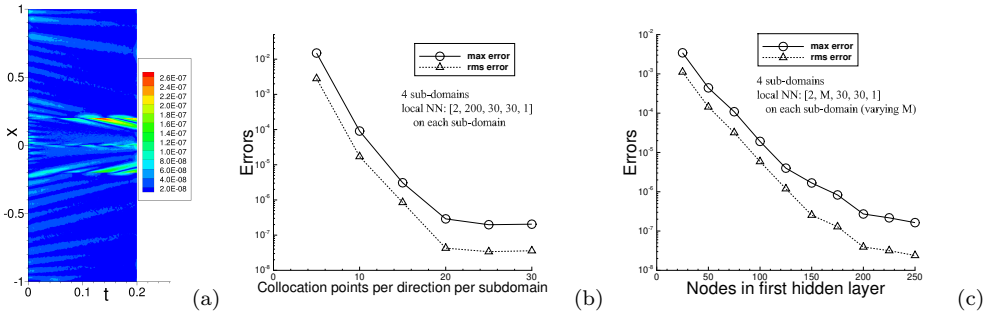

Fig. 19 Burgers' equation on the smaller domain Ω_1 (3 hidden layers in network): (a) Error distribution of the locHLConcELM solution. The locHLConcELM maximum/rms errors versus (b) the number of collocation points per direction in each sub-domain, and (c) the number of nodes in the first hidden layer (M). Local network architecture: $[2, M, 30, 30, 1]$. $M = 200$ in (a,b), and is varied in (c). $Q = 21 \times 21$ in (a,c), and is varied in (b). $\mathbf{R} = (1.0, 0.035, 0.03)$ in (a,b,c).

smaller domain Ω_1. With both methods, we employ 4 sub-domains as shown in Figure 16(a), a local neural network architecture $[2, 200, 30, 1]$, and a sequence of uniform collocation points ranging from $Q = 5 \times 5$ and $Q = 30 \times 30$. The current locHLConcELM method is significantly more accurate than the conventional locELM method, with their maximum errors on the order of 10^{-7} and 10^{-2} respectively.

Figure 19 illustrates the characteristics of the locHLConcELM solution obtained with 3 hidden layers in the local neural network on the smaller domain Ω_1. Here we employ a local network architecture $[2, M, 30, 30, 1]$, with M either fixed at $M = 200$ or varied systematically, and a uniform set of $Q = Q_1 \times Q_1$ collocation points, with Q_1 either fixed at $Q_1 = 21$ or varied systematically. The hidden magnitude vector is $\mathbf{R} = (1.0, 0.035, 0.03)$, obtained using the method of [13]. Figure 19(a) is an illustration of the absolute-error distribution on Ω_1 corresponding to $M = 200$ and $Q_1 = 21$, demonstrating a high accuracy with the maximum error on the order of 10^{-7}. Figures 19(b) and (c) show the exponential convergence behavior of the HLConcELM errors with respect to Q_1 and M.

Let us next consider the larger domain Ω ($t \in [0, 1]$) and solve the system (27) using the current method. We employ the locHLConcELM method together with the block time marching scheme (see Remarks 5 and 6) in the simulation. Specif-

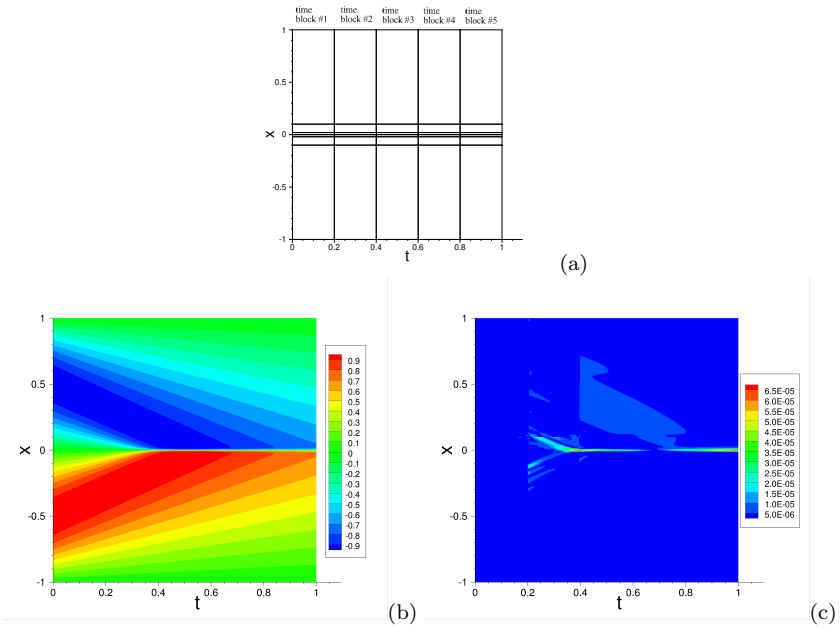

Fig. 20 Burgers' equation on the larger domain Ω ($t \in [0,1]$): (a) Configuration of the 5 uniform time blocks and the 6 non-uniform sub-domains on each time block. Distributions of (b) the locHLConcELM solution and (c) its absolute error. Local network architecture $[2, 300, 1]$ on each sub-domain, $Q = 21 \times 21$ per sub-domain, $\mathbf{R} = 2.0$.

ically, we divide the temporal dimension into 5 uniform time blocks, and partition each time block into 6 non-uniform sub-domains along the x direction. Figure 20(a) illustrates the configuration of the time blocks and the sub-domains on each time block, where the x coordinates of the sub-domain boundaries are given by $\boldsymbol{\mathcal{X}} = [-1, -0.1, -0.02, 0, 0.02, 0.1, 1]$. We employ a local neural network architecture $\mathbf{M} = [2, 300, 1]$ and a uniform set of $Q = 21 \times 21$ collocation points on each sub-domain. The hidden magnitude vector is $\mathbf{R} = 2.0$, which is obtained using the method of [13]. Figures 20(b) and (c) show the distributions of the locHLConcELM solution and its absolute error on Ω. The data indicate that the current method achieves a quite high accuracy with the sharp gradient present in the domain, with the maximum error on the order of 10^{-5}.

Figure 21 compares profiles of the locHLConcELM solution and the exact solution (28) for the Burgers' equation at three time instants $t = 0.25$, 0.5 and 1.0. The error profiles of the locHLConcELM solution have also been included in this figure. The simulation configuration and the parameters used here correspond to those of Figure 20. It is evident that the current locHLConcELM method has achieved a quite high accuracy for this problem.

3.5 KdV Equation

In the next benchmark problem we employ the Korteweg-de Vries (KdV) equation to test the HLConcELM method. Consider the spatial-temporal domain $(x, t) \in$

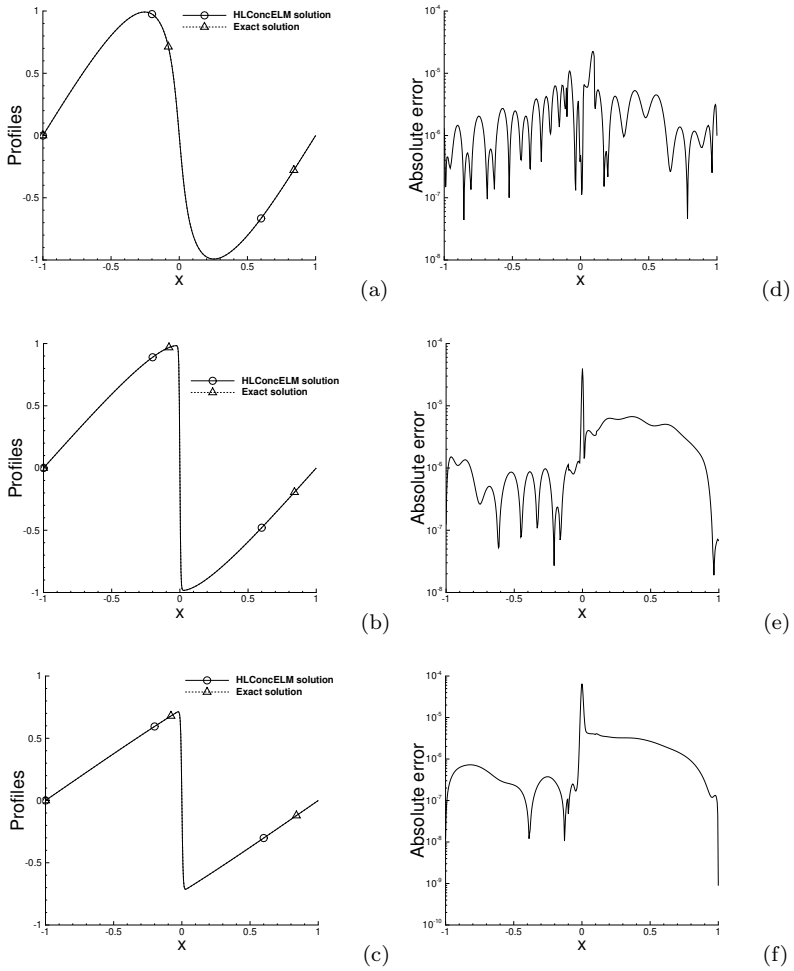

Fig. 21 Burgers' equation on the larger domain Ω: Profiles of the locHLConcELM solution (first column) and its absolute error (second column) at the time instants: $t = 0.25$ (a,d), $t = 0.5$ (b,e), and $t = 1.0$ (c,f). The profiles of the exact solution are also shown in (a,b,c), which overlap with those of the locHLConcELM solution. The locHLConcELM simulation parameters and configurations follow those of Figure 20.

$\Omega = [1.0, 1.5] \times [1.0, 1.5]$ and the following initial/boundary value problem,

$$\frac{\partial u}{\partial t} - u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = f(x,t), \tag{29a}$$

$$u(1,t) = g_1(t), \quad u(1.5,t) = g_2(t), \quad \frac{\partial u}{\partial x}(1,t) = g_3(t), \tag{29b}$$

$$u(x,1) = h(x). \tag{29c}$$

In the above equations $u(x,t)$ is the field solution to be sought, $f(x,t)$ is a prescribed source term, g_i $(i = 1, 2, 3)$ and h are the data for the boundary and initial

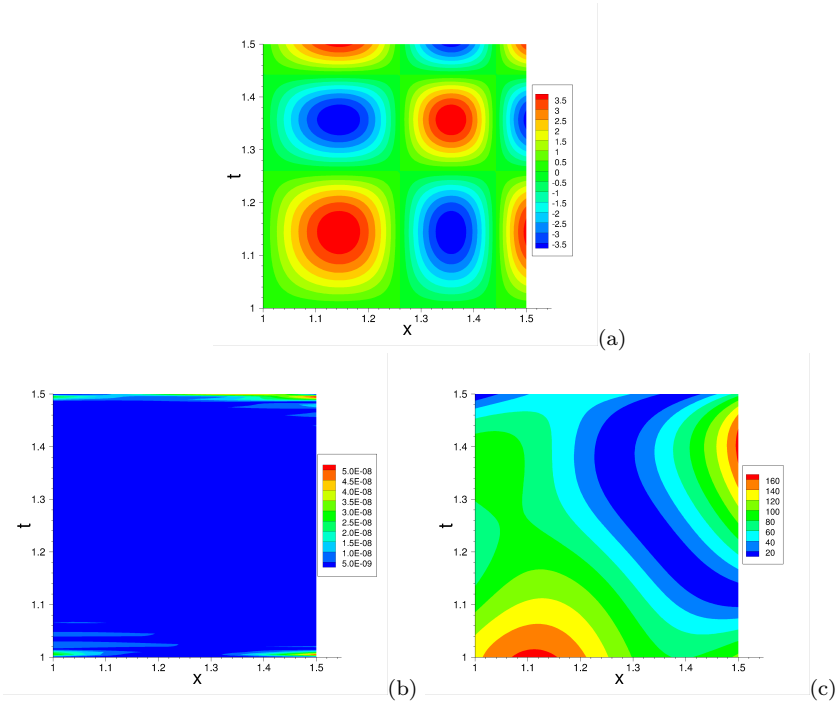

Fig. 22 KdV equation: Distributions of (a) the exact solution, (b) the absolute error of the HLConcELM solution, and (c) the absolute error of the conventional ELM solution. In (b,c), neural network architecture $\mathbf{M} = [2, 800, 50, 1]$, $Q = 35 \times 35$ uniform collocation points. $\mathbf{R} = (3.2, 0.01)$ in (b) for HLConcELM. $R_m = R_{m0} = 0.27$ in (c) for conventional ELM.

conditions. We choose f, g_i $(i = 1, 2, 3)$ and h such that the system (29) has the following analytic solution,

$$u(x,t) = 4 \sin\left(\pi x^3\right) \sin\left(\pi t^3\right). \tag{30}$$

Figure 22(a) shows the distribution of this solution in the spatial-temporal xt plane.

To solve the problem (29) using the HLConcELM method, we employ neural networks with two input nodes (representing the x and t) and a single output node (representing u), with the Gaussian activation function for all the hidden nodes. A uniform set of $Q = Q_1 \times Q_1$ collocation points on the domain Ω is used to train the neural network, where Q_1 is varied systematically in the tests.

Figures 22(b) and (c) illustrate the absolute-error distributions obtained using the HLConcELM method and the conventional ELM method. Here the network architecture is given by $\mathbf{M} = [2, 800, 50, 1]$, and a uniform set of $Q = 35 \times 35$ collocation points is used for both methods. The hidden magnitude vector is $\mathbf{R} = (3.2, 0.01)$ with HLConcELM, which is obtained using the method of [13]. For conventional ELM, we set the hidden-layer coefficients to random values generated on $[-R_m, R_m]$ with $R_m = R_{m0}$, where $R_{m0} = 0.27$ is the optimal R_m obtained using the method of [13]. The conventional ELM solution is observed to be inaccurate (maximum error on the order of 10^2), because of the narrow

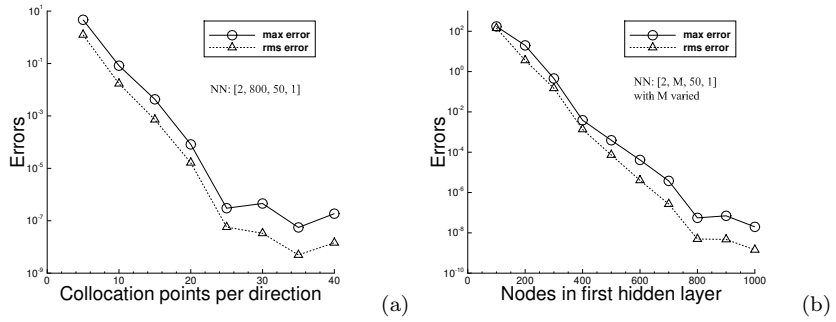

Fig. 23 KdV equation: The HLConcELM maximum/rms errors versus (a) the number of collocation points per direction, and (b) the number of nodes in the first hidden layer (M). Network architecture $[2, M, 50, 1]$. $M = 800$ in (a), varied in (b). $Q = 35 \times 35$ in (b), varied in (a). $\mathbf{R} = (3.2, 0.01)$ in (a,b).

neural network	collocation points	current max error	HLConcELM rms error	conventional max error	ELM rms error
$[2, 800, 50, 1]$	5×5	$4.67E+0$	$1.24E+0$	$1.66E+2$	$8.18E+1$
	10×10	$8.28E-2$	$1.70E-2$	$1.78E+2$	$1.02E+2$
	15×15	$4.30E-3$	$7.21E-4$	$1.88E+2$	$9.36E+1$
	20×20	$8.21E-5$	$1.64E-5$	$1.78E+2$	$9.80E+1$
	25×25	$3.02E-7$	$5.71E-8$	$1.75E+2$	$8.89E+1$
	30×30	$4.55E-7$	$3.32E-8$	$1.66E+2$	$7.76E+1$
	35×35	$5.55E-8$	$4.95E-9$	$1.67E+2$	$7.77E+1$
	40×40	$1.87E-7$	$1.45E-8$	$1.67E+2$	$7.74E+1$

Table 6 KdV equation: Comparison of the maximum/rms errors from the HLConcELM method and the conventional ELM method. Network architecture: $[2, 800, 50, 1]$. The HLConcELM data in this table correspond to those in Figure 23(a). For conventional ELM, the hidden-layer coefficients are set to uniform random values generated on $[-R_m, R_m]$ with $R_m = R_{m0}$. Here $R_{m0} = 0.27$ is the optimal R_m obtained using the method of [13].

last hidden layer in the network. In contrast, the HLConcELM solution is highly accurate, with the maximum error on the order of 10^{-8} in the domain.

Figure 23 illustrates the convergence behavior of the HLConcELM errors with respect to the collocation points and the number of nodes in the network. Here the network architecture is $\mathbf{M} = [2, M, 50, 1]$, with M either fixed at $M = 800$ or varied between $M = 100$ and $M = 1000$. A uniform set of $Q = Q_1 \times Q_1$ collocation points is used, with Q_1 either fixed at $Q_1 = 35$ or varied between $Q_1 = 5$ and $Q_1 = 40$. The hidden magnitude vector is $\mathbf{R} = (3.2, 0.01)$, obtained from the method of [13]. The two plots (a) and (b) depict the maximum/rms errors of the HLConcELM solution as a function of Q_1 and M, respectively. One can observe the familiar exponential decrease in the errors with increasing Q_1 or M.

Table 6 provides an accuracy comparison of the HLConcELM method and the conventional ELM method [9] for solving the KdV equation on a network architecture $\mathbf{M} = [2, 800, 50, 1]$ corresponding to a sequence of collocation points. The HLConcELM solution is highly accurate, while the conventional ELM solution exhibits no accuracy at all on such a neural network.

Figure 24 illustrates the HLConcELM solutions obtained on a deeper neural network with 3 hidden layers. The neural network architecture is given by $\mathbf{M} =$

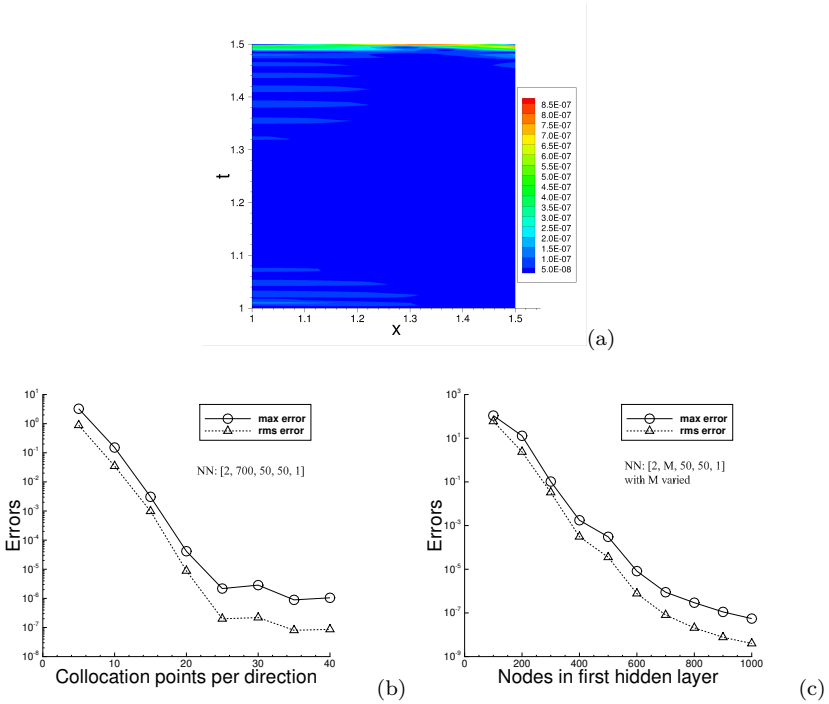

Fig. 24 KdV equation (3 hidden layers in network): (a) Distribution of the absolute error of the HLConcELM solution. The HLConcELM maximum/rms errors versus (b) the number of collocation points per direction, and (c) the number of nodes in the first hidden layer (M). Network architecture: $[2, M, 50, 50, 1]$. $M = 700$ in (a,b), varied in (c). $Q = 35 \times 35$ in (a,c), varied in (b). $\mathbf{R} = (3.0, 0.025, 1.6)$ in (a,b,c).

$[2, M, 50, 50, 1]$, where M is either fixed at $M = 700$ or varied systematically. A hidden magnitude vector $\mathbf{R} = (3.0, 0.025, 1.6)$ is employed in all the simulations. Figure 24(a) shows the absolute-error distribution on Ω, which corresponds to $M = 700$ and a set of $Q = 35 \times 35$ collocation points. The HLConcELM result is highly accurate, with the maximum error on the order of 10^{-7} on Ω. Figures 24(b) and (c) depict the maximum/rms errors of HLConcELM as a function of the number of collocation points and as a function of M, respectively. The data again signify the exponential (or near exponential) convergence of the HLConcELM errors.

4 Concluding Remarks

The extreme learning machine (ELM) method can yield highly accurate solutions to linear and nonlinear PDEs. To achieve a high accuracy, the existing ELM method [9,10,13] requires the last hidden layer of the neural network to be wide. So the ELM neural network typically contains a large number of nodes in the last hidden layer, irrespective of the rest of the network configuration. If the last hidden layer is narrow, the ELM accuracy will suffer and tend to be poor, even

though the neural network may contain a large number of the nodes in the other hidden layers.

In the current paper we have presented a method to overcome the above drawback of the existing (conventional) ELM method. The new method, termed HLConcELM (hidden-layer concatenated ELM), can produce highly accurate solutions to PDEs when the last hidden layer is wide, and when the last hidden layer is narrow, in which case the conventional ELM completely losses accuracy.

The new method relies on a type of modified feedforward neural networks (FNN), which exposes the hidden nodes in all the hidden layers to the output nodes by incorporating a *logical* concatenation of the hidden layers into the network. We refer to this modified network as HLConcFNN (hidden-layer concatenated FNN) in this paper. In HLConcFNN every hidden node in the network directly influences the nodes in the output layer, while in conventional FNN only the hidden nodes in the last hidden layer directly influence the output nodes. HLConcFNNs have the property that, if new hidden layers are appended to an existing network architecture or new nodes are added to an existing hidden layer, the representation capacity of the resultant network architecture is guaranteed to be not smaller than that of the original one.

The HLConcELM method adopts the HLConcFNN as its neural network. It assigns random values to (and fixes) the weight/bias coefficients in all the hidden layers of the neural network, while the coefficients between the output nodes and all the hidden nodes of the network are trained/computed by a linear or nonlinear least squares method. Note that in HLConcELM every hidden node in the network is connected to the output nodes because of the logical hidden-layer concatenation. HLConcELMs partially inherit the non-decreasing representation capacity property of HLConcFNNs. They have the property that, as new hidden layers are appended to an existing network architecture, the representation capacity of the HLConcELM associated with the resultant architecture is not smaller than that associated with the original one, provided that the random coefficients in the resultant architecture are assigned in an appropriate fashion.

In essence, when solving PDEs or approximating functions, the ELM method performs an expansion of the unknown field solution in terms of a set of random basis functions. With conventional ELM, the random bases consist of the output fields of the last hidden layer of the neural network. With HLConcELM, on the other hand, the random bases consist of the output fields of the hidden nodes in all the hidden layers of the neural network. HLConcELM is able to harvest the degrees of freedom provided by all the hidden nodes in the network, not limited to those from the last hidden layer. This is the essential difference between the HLConcELM method and the conventional ELM method.

We have tested the current HLConcELM method on boundary value problems and initial/boundary value problems involving a number of linear and nonlinear PDEs. In particular, we have compared HLConcELM and the conventional ELM on network architectures whose last hidden layer is narrow or wide. The numerical results demonstrate that the current HLConcELM method yields highly accurate results on network architectures with both narrow and wide last hidden layers. In contrast, the conventional ELM only achieves accurate results on architectures with a wide last hidden layer, and with a narrow last hidden layer it exhibits poor or no accuracy. The HLConcELM method displays an exponentially convergent behavior for smooth field solutions, reminiscent of the traditional high-order

techniques [35,72,69,70,68]. Its numerical errors decrease exponentially or nearly exponentially as the number of collocation points or the number of trainable parameters increases.

Appendix A: Proofs of Theorems from Section 2

Proof of Theorem 1:

Consider an arbitrary $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x}) \in U(\Omega,\mathbf{M}_1,\sigma)$, where $\boldsymbol{\theta} \in \mathbb{R}^{N_{h1}}$ and $\boldsymbol{\beta} \in \mathbb{R}^{N_{c1}}$, with $N_{h1} = \sum_{i=1}^{L-1}(m_{i-1}+1)m_i$ and $N_{c1} = \sum_{i=1}^{L-1} m_i$. Let $w_{kj}^{(i)}$ $(1 \leqslant i \leqslant L-1,\ 1 \leqslant k \leqslant m_{i-1},\ 1 \leqslant j \leqslant m_i)$ and $b_j^{(i)}$ $(1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i)$ denote the hidden-layer weight/bias coefficients of the associated HLConcFNN$(\mathbf{M}_1,\sigma)$, and let β_{ij} $(1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i)$ denote the output-layer coefficients of HLConcFNN$(\mathbf{M}_1,\sigma)$. $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x})$ is given by (7).

Consider a function $v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x}) \in U(\Omega,\mathbf{M}_2,\sigma)$ with $\boldsymbol{\vartheta} \in \mathbb{R}^{N_{h2}}$ and $\boldsymbol{\alpha} \in \mathbb{R}^{N_{c2}}$, where $N_{c2} = N_{c1}+n$, and $N_{h2} = N_{h1}+(m_{L-1}+1)n$. We will choose $\boldsymbol{\vartheta}$ and $\boldsymbol{\alpha}$ such that $v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x}) = u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x})$. We construct $\boldsymbol{\vartheta}$ and $\boldsymbol{\alpha}$ by setting the hidden-layer and the output-layer coefficients of HLConcFNN$(\mathbf{M}_2,\sigma)$ as follows.

The HLConcFNN$(\mathbf{M}_2,\sigma)$ has L hidden layers. We set the weight/bias coefficients in its last hidden layer (with n nodes) to arbitrary values. We set those coefficients that connect the output node and the n nodes in the last hidden layer to all zeros. For the rest of the hidden-layer coefficients and the output-layer coefficients in HLConcFNN$(\mathbf{M}_2,\sigma)$, we use those corresponding coefficient values from the network HLConcFNN$(\mathbf{M}_1,\sigma)$.

More specifically, let $\xi_{kj}^{(i)}$ and $\eta_j^{(i)}$ denote the weight/bias coefficients in the hidden layers, and α_{ij} denote the output-layer coefficients, of HLConcFNN$(\mathbf{M}_2,\sigma)$ associated with the function $v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x})$. We set these coefficients by,

$$\xi_{kj}^{(i)} = \begin{cases} w_{kj}^{(i)}, & \text{for } 1 \leqslant i \leqslant L-1,\ 1 \leqslant k \leqslant m_{i-1},\ 1 \leqslant j \leqslant m_i; \\ \text{arbitrary value, for } i = L,\ 1 \leqslant k \leqslant m_{L-1},\ 1 \leqslant j \leqslant n; \end{cases} \tag{31}$$

$$\eta_j^{(i)} = \begin{cases} b_j^{(i)}, & \text{for all } 1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i; \\ \text{arbitrary value, for } i = L,\ 1 \leqslant j \leqslant n; \end{cases} \tag{32}$$

$$\alpha_{ij} = \begin{cases} \beta_{ij}, & \text{for } 1 \leqslant i \leqslant L-1,\ 1 \leqslant j \leqslant m_i; \\ 0, & \text{for } i = L,\ 1 \leqslant j \leqslant n. \end{cases} \tag{33}$$

With the above coefficients, the last hidden layer of the network HLConcFNN$(\mathbf{M}_2,\sigma)$ may output arbitrary fields, which however have no effect on the output field of HLConcFNN$(\mathbf{M}_2,\sigma)$ because $\alpha_{Lj} = 0$ $(1 \leqslant j \leqslant n)$. The rest of the hidden nodes in HLConcFNN$(\mathbf{M}_2,\sigma)$ and the output node of HLConcFNN$(\mathbf{M}_2,\sigma)$ produce fields that are identical to those of the corresponding nodes in the network HLConcFNN$(\mathbf{M}_1,\sigma)$. We thus conclude that $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x}) = v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x})$. So $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x}) \in U(\Omega,\mathbf{M}_2,\sigma)$, and the relation (9) holds.

Proof of Theorem 2:

We use the same strategy as that in the proof of Theorem 1. Consider an arbitrary $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x}) \in U(\Omega,\mathbf{M}_1,\sigma)$, where $\boldsymbol{\theta} \in \mathbb{R}^{N_{h1}}$ and $\boldsymbol{\beta} \in \mathbb{R}^{N_{c1}}$, with $N_{h1} =$

$\sum_{i=1}^{L-1}(m_{i-1}+1)m_i$ and $N_{c1}=\sum_{i=1}^{L-1}m_i$. The hidden-layer coefficients of the associated HLConcFNN$(\mathbf{M}_1,\sigma)$ are denoted by $w_{kj}^{(i)}$ $(1\leqslant i\leqslant L-1,\ 1\leqslant k\leqslant m_{i-1},\ 1\leqslant j\leqslant m_i)$ and $b_j^{(i)}$ $(1\leqslant i\leqslant L-1,\ 1\leqslant j\leqslant m_i)$, and the output-layer coefficients are denoted by β_{ij} $(1\leqslant i\leqslant L-1,\ 1\leqslant j\leqslant m_i)$. $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x})$ is given by (7).

Consider a function $v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x})\in U(\Omega,\mathbf{M}_2,\sigma)$ with $\boldsymbol{\vartheta}\in\mathbb{R}^{N_{h2}}$ and $\boldsymbol{\alpha}\in\mathbb{R}^{N_{c2}}$, where $N_{c2}=N_{c1}+1$, and $N_{h2}=N_{h1}+(m_{s-1}+1)+m_{s+1}$ if $1\leqslant s\leqslant L-2$ and $N_{h2}=N_{h1}+(m_{s-1}+1)$ if $s=L-1$. We construct $\boldsymbol{\vartheta}$ and $\boldsymbol{\alpha}$ by setting the hidden-layer and the output-layer coefficients of HLConcFNN$(\mathbf{M}_2,\sigma)$ as follows.

In HLConcFNN$(\mathbf{M}_2,\sigma)$ we set the weight coefficients that connect the extra node of layer s to those nodes in layer $(s+1)$ to all zeros, and we also set the weight coefficient that connects the extra node of layer s with the output node to zero. We set the weight coefficients that connect the nodes of layer $(s-1)$ to the extra node of layer s to arbitrary values, and also set the bias coefficient corresponding to the extra node of layer s to an arbitrary value. For the rest of the hidden-layer and output-layer coefficients of HLConcFNN$(\mathbf{M}_2,\sigma)$, we use those corresponding coefficient values from the network HLConcFNN$(\mathbf{M}_1,\sigma)$.

Specifically, let $\xi_{kj}^{(i)}$ and $\eta_j^{(i)}$ denote the weight/bias coefficients in the hidden layers, and α_{ij} denote the output-layer coefficients, of the HLConcFNN$(\mathbf{M}_2,\sigma)$ associated with $v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x})$. We set these coefficients by,

$$\xi_{kj}^{(i)}=\begin{cases} w_{kj}^{(i)}, & \text{for all } (1\leqslant i\leqslant s-1,\ \text{or } s+2\leqslant i\leqslant L-1),\\ & \quad 1\leqslant k\leqslant m_{i-1},\ 1\leqslant j\leqslant m_i;\\ w_{kj}^{(s)}, & \text{for } i=s,\ 1\leqslant k\leqslant m_{s-1},\ 1\leqslant j\leqslant m_s;\\ \text{arbitrary value}, & \text{for } i=s,\ 1\leqslant k\leqslant m_{s-1},\ j=m_s+1;\\ w_{kj}^{(s+1)}, & \text{for } i=s+1,\ 1\leqslant k\leqslant m_s,\ 1\leqslant j\leqslant m_{s+1};\\ 0, & \text{for } i=s+1,\ k=m_s+1,\ 1\leqslant j\leqslant m_{s+1};\end{cases} \tag{34}$$

$$\eta_j^{(i)}=\begin{cases} b_j^{(i)}, & \text{for all } 1\leqslant i\leqslant L-1,\ i\neq s,\ 1\leqslant j\leqslant m_i;\\ b_j^{(s)}, & \text{for } i=s,\ 1\leqslant j\leqslant m_s;\\ \text{arbitrary value}, & \text{for } i=s,\ j=m_s+1;\end{cases} \tag{35}$$

$$\alpha_{ij}=\begin{cases} \beta_{ij}, & \text{for all } 1\leqslant i\leqslant L-1,\ i\neq s,\ 1\leqslant j\leqslant m_i;\\ \beta_{sj}, & \text{for } i=s,\ 1\leqslant j\leqslant m_s;\\ 0, & \text{for } i=s,\ j=m_s+1.\end{cases} \tag{36}$$

With the above coefficients, the extra node in layer s of the network HLConcFNN$(\mathbf{M}_2,\sigma)$ may output an arbitrary field, which however has no contribution to the output field of HLConcFNN$(\mathbf{M}_2,\sigma)$. The rest of the hidden nodes and the output node of HLConcFNN$(\mathbf{M}_2,\sigma)$ produce identical fields as the corresponding nodes in the network HLConcFNN$(\mathbf{M}_1,\sigma)$. We thus conclude that $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x})=v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x})$. So $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x})\in U(\Omega,\mathbf{M}_2,\sigma)$ and the relation (10) holds.

Proof of Theorem 3:

We use the same strategy as that in the proof of Theorem 1. Consider an arbitrary $u(\boldsymbol{\theta},\boldsymbol{\beta},\mathbf{x})\in U(\Omega,\mathbf{M}_1,\sigma,\boldsymbol{\theta})$, where $\boldsymbol{\beta}\in\mathbb{R}^{N_{c1}}$ with $N_{c1}=\sum_{i=1}^{L-1}m_i$. We will try to construct an equivalent function from $U(\Omega,\mathbf{M}_2,\sigma,\boldsymbol{\vartheta})$.

We consider another function $v(\boldsymbol{\vartheta},\boldsymbol{\alpha},\mathbf{x})\in U(\Omega,\mathbf{M}_2,\sigma,\boldsymbol{\vartheta})$, where $\boldsymbol{\alpha}\in\mathbb{R}^{N_{c2}}$ with $N_{c2}=N_{c1}+n$, and we set the coefficients of the HLConcELM corresponding to

name	$\sigma(x)$	$\mathbf{R}$	name	$\sigma(x)$	$\mathbf{R}$
Gaussian	e^{-x^2}	(3.0,0.005)	sinc	$\frac{\sin(\pi x)}{\pi x}$	(6.0,0.05)
tanh	$\tanh(x)$	(2.0,0.05)	GELU	$\frac{1}{2}\left(1+\mathrm{erf}\frac{x}{\sqrt{2}}\right)$	(4.4,0.01)
RePU-8	$\begin{cases} x^8, & if\ x \geqslant 0, \\ 0, & if\ x < 0. \end{cases}$	(0.22,0.4)	swish	$\frac{x}{1+e^{-x}}$	(3.8,0.01)

Table 7 Appendix B (variable-coefficient Poisson equation): The activation functions and the corresponding hidden magnitude vector $\mathbf{R}$ employed. The $\mathbf{R}$ values used here are close to the optimal $\mathbf{R}^*$ obtained by the method of [13] based on the neural network [2,800,50,1] with Q=35×35 collocation points.

$\sigma(x)$	max	error	rms	error	h^1	error
	Q=15×15	30×30	Q=15×15	30×30	Q=15×15	30×30
Gaussian	2.33E-3	3.17E-8	3.92E-4	3.79E-9	8.15E-3	1.33E-7
tanh	2.33E-3	1.64E-6	3.14E-4	1.10E-7	6.30E-3	5.28E-6
RePU-8	3.40E-3	8.19E-3	6.66E-4	7.30E-4	1.08E-2	1.76E-2
sinc	2.35E-2	4.37E-9	8.27E-3	3.35E-10	1.16E-1	1.51E-8
GELU	2.89E-3	8.75E-8	4.55E-4	5.73E-9	9.34E-3	2.93E-7
swish	1.31E-3	7.07E-7	2.43E-4	5.43E-8	4.95E-3	2.59E-6

Table 8 Appendix B (variable-coefficient Poisson equation): The max/rms/h^1 errors of HLConcELM obtained with different activation functions on two uniform sets of collocation points. Neural network [2,800,50,1]. $\mathbf{R}$ values are given in Table 7.

$v(\boldsymbol{\vartheta}, \boldsymbol{\alpha}, \mathbf{x})$ as follows. Since $\boldsymbol{\vartheta}[1 : N_{h1}] = \boldsymbol{\theta}[1 : N_{h1}]$, the random coefficients in the first $(L-1)$ hidden layers of the HLConcELM corresponding to $v(\boldsymbol{\vartheta}, \boldsymbol{\alpha}, \mathbf{x})$ are identical to those corresponding hidden-layer coefficients in the HLConcELM for $u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x})$. We set the weight/bias coefficients in the L-th hidden layer of the HLConcELM for $v(\boldsymbol{\vartheta}, \boldsymbol{\alpha}, \mathbf{x})$, which contains n nodes, to arbitrary random values. For the output-layer coefficients of the HLConcELM for $v(\boldsymbol{\vartheta}, \boldsymbol{\alpha}, \mathbf{x})$, we set those coefficients that connect the hidden nodes in the first $(L-1)$ hidden layers and the output node to be identical to those corresponding output-layer coefficients in the HLConcELM for $u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x})$, namely, $\boldsymbol{\alpha}[1 : N_{c1}] = \boldsymbol{\beta}[1 : N_{c1}]$. We set those coefficients that connect the hidden nodes of the L-th hidden layer and the output node to be zeros in the HLConcELM for $v(\boldsymbol{\vartheta}, \boldsymbol{\alpha}, \mathbf{x})$, namely, $\boldsymbol{\alpha}[N_{c1}+1 : N_{c2}] = 0$.

With the above coefficient settings, the output fields of those nodes in the first $(L-1)$ hidden layers of HLConcELM$(\mathbf{M}_2, \sigma, \boldsymbol{\vartheta})$ are identical to those corresponding nodes of HLConcELM$(\mathbf{M}_1, \sigma, \boldsymbol{\theta})$. The output fields of those n nodes in the L-th hidden layer of HLConcELM$(\mathbf{M}_2, \sigma, \boldsymbol{\vartheta})$ are arbitrary, which however have no contribution to the output field of HLConcELM$(\mathbf{M}_2, \sigma, \boldsymbol{\vartheta})$. The output field of the HLConcELM$(\mathbf{M}_2, \sigma, \boldsymbol{\vartheta})$ is identical to that of the HLConcELM$(\mathbf{M}_1, \sigma, \boldsymbol{\theta})$, i.e. $v(\boldsymbol{\vartheta}, \boldsymbol{\alpha}, \mathbf{x}) = u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x})$. We thus conclude that $u(\boldsymbol{\theta}, \boldsymbol{\beta}, \mathbf{x}) \in U(\Omega, \mathbf{M}_2, \sigma, \boldsymbol{\vartheta})$ and the relation (13) holds.

Appendix B. Numerical Tests with Several Activation Functions

We have employed the Gaussian activation function for all the numerical simulations in Section 3. This appendix provides additional HLConcELM results using several other activation functions for solving the variable-coefficient Poisson problem from Section 3.1. Table 7 lists the activation functions studied below, including

$\sigma(x)$	max	error	rms	error	h^1	error
	M=400	800	M=400	800	M=400	800
Gaussian	9.00E-4	5.68E-8	7.96E-5	5.25E-9	2.26E-3	2.42E-7
tanh	1.84E-3	1.44E-6	2.06E-4	1.12E-7	6.34E-3	5.39E-6
RePU-8	1.19E-1	6.61E-3	1.20E-2	5.47E-4	2.32E-1	1.38E-2
sinc	1.32E-3	3.33E-9	1.51E-4	1.61E-10	4.06E-3	1.38E-8
GELU	1.06E-3	1.05E-7	7.71E-5	7.23E-9	2.24E-3	3.77E-7
swish	1.50E-3	1.78E-6	1.17E-4	7.17E-8	3.58E-3	4.35E-6

Table 9 Appendix B (variable-coefficient Poisson equation): The max/rms/h^1 errors of HLConcELM obtained with different activation functions on two neural networks with architecture $[2, M, 50, 1]$. $Q = 35 \times 35$ uniform collocation points. $\mathbf{R}$ values are given in Table 7.

tanh, RePU-8, sinc, GELU and swish (in addition to Gaussian), as well as the hidden magnitude vector $\mathbf{R}$ employed for each activation function. Here "RePU-8" stands for the rectified power unit of degree 8, and "GELU" denotes the Gaussian error linear unit [25].

Table 8 lists the maximum, rms and h^1 errors of the HLConcELM solutions obtained using these activation functions on a neural network $[2, 800, 50, 1]$ with two uniform sets of collocation points $Q = 15 \times 15$ and 30×30. Table 9 lists the maximum, rms and h^1 errors of HLConcELM using these activation functions on two neural networks of the architecture $[2, M, 50, 1]$ (with $M = 400$ and 800) with a fixed uniform set of $Q = 35 \times 35$ collocation points. One can observe a general exponential decrease in the errors with these activation functions, except for the RePU-8 function in Table 8 (where the errors seem to saturate). The results with the RePU function appears markedly less accurate than those obtained with the other activation functions studied here.

Appendix C. Additional Comparisons Between HLConcELM and Conventional ELM

This appendix provides additional comparisons between the current HLConcELM method and the conventional ELM method for the variable-coefficient Poisson problem (Section 3.1) and the nonlinear Helmholtz problem (Section 3.3).

In those comparisons between HLConcELM and conventional ELM presented in Section 3, the base neural-network architectures for HLConcELM and conventional ELM are maintained to be the same. HLConcELM is able to harvest the degrees of freedom in all the hidden layers of the neural network, thanks to the logical connections between all the hidden nodes and the output nodes (due to the hidden-layer concatenation). On the other hand, the conventional ELM only exploits the degrees of freedom afforded by the last hidden layer of the network, while those degrees of freedom provided by the preceding hidden layers are essentially "wasted" (see the discussions in Section 2.1). This is why the conventional ELM exhibits a poor accuracy if the last hidden layer is narrow, irrespective of the rest of the network configuration. This also accounts for why the HLConcELM method can achieve a high accuracy when the last hidden layer is narrow and when it is wide.

Note that with HLConcELM the number of training parameters equals the total number of hidden nodes in the neural network, and with conventional ELM it

method	network	Q	max error	rms error
HLConcELM	$[2, 800, 50, 1]$	5×5	1.91E+0	4.31E-1
		10×10	3.22E-2	7.88E-3
		15×15	2.33E-3	3.92E-4
		20×20	4.70E-5	1.32E-5
		25×25	4.78E-7	1.10E-7
		30×30	3.17E-8	3.79E-9
Conventional ELM	$[2, 800, 850, 1]$	5×5	2.34E+0	7.43E-1
		10×10	1.02E-1	1.98E-2
		15×15	3.58E-3	9.09E-4
		20×20	9.82E-5	2.35E-5
		25×25	2.97E-6	2.76E-7
		30×30	4.49E-6	3.26E-7
HLConcELM	$[2, 50, 800, 1]$	5×5	2.95E+0	9.26E-1
		10×10	9.35E-2	9.93E-3
		15×15	1.11E-3	2.40E-4
		20×20	3.42E-5	6.91E-6
		25×25	2.34E-6	4.45E-7
		30×30	3.07E-8	4.81E-9
Conventional ELM	$[2, 50, 850, 1]$	5×5	3.15E+0	1.05E+0
		10×10	1.16E-1	1.92E-2
		15×15	3.88E-3	7.55E-4
		20×20	5.59E-5	1.91E-5
		25×25	8.60E-7	2.16E-7
		30×30	6.89E-8	1.14E-8

Table 10 Appendix C (variable-coefficient Poisson equation): Comparison of the maximum and rms errors versus the number of collocation points (Q) obtained by HLConcELM and conventional ELM. In all cases, the neural network has two hidden layers and a total of 850 trainable parameters for both HLConcELM and conventional ELM. The HLConcELM data in this table correspond to those in Table 1. For conventional ELM, the hidden-layer coefficients are assigned to uniform random values from $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m obtained using the method of [13], with $R_{m0} = 0.38$ for the network $[2, 800, 850, 1]$ and $R_{m0} = 0.72$ for the network $[2, 50, 850, 1]$.

equals the number of nodes in the last hidden layer. Under the same base network architecture (with multiple hidden layers), the number of training parameters in HLConcELM is larger than that in the conventional ELM, because the HLConcELM also exploits the the hidden nodes from the preceding hidden layers.

In what follows we present several additional numerical tests to compare HLConcELM and conventional ELM, under the configuration that the number of training parameters in both HLConcELM and conventional ELM is maintained to be the same. Because of their different characteristics, the base network architectures for HLConcELM and for conventional ELM in this case will inevitably not be identical. In the comparisons below we try to keep the two architectures close to each other, specifically by using the same depth, and the same width for each hidden layer except the last, for both HLConcELM and conventional ELM. The width of the last hidden layer in the HLConcELM network and in the conventional-ELM network is different, with the conventional ELM being wider (and in some cases considerably wider), while the number of training parameters is kept the same in both.

Tables 10 and 11 show comparisons of the maximum and rms errors versus the number of collocation points obtained by HLConcELM and by conventional ELM for the variable-coefficient Poisson problem from Section 3.1. The results in

method	network	Q	max error	rms error
HLConcELM	$[2, 800, 50, 50, 1]$	5×5	1.92E+0	4.35E-1
		10×10	3.21E-2	7.91E-3
		15×15	2.33E-3	3.94E-4
		20×20	6.07E-5	1.45E-5
		25×25	4.99E-7	1.16E-7
		30×30	6.44E-8	5.62E-9
Conventional ELM	$[2, 800, 50, 900, 1]$	5×5	2.83E+0	8.32E-1
		10×10	2.52E-1	4.58E-2
		15×15	7.92E-3	1.16E-3
		20×20	1.14E-4	1.32E-5
		25×25	4.31E-6	8.18E-7
		30×30	1.36E-6	9.22E-8
HLConcELM	$[2, 50, 50, 800, 1]$	5×5	1.81E+0	4.40E-1
		10×10	5.83E-2	1.11E-2
		15×15	3.91E-3	6.24E-4
		20×20	4.57E-5	6.39E-6
		25×25	8.10E-7	1.23E-7
		30×30	2.97E-8	2.66E-9
Conventional ELM	$[2, 50, 50, 900, 1]$	5×5	1.49E+0	4.21E-1
		10×10	8.79E-2	2.06E-2
		15×15	1.90E-3	3.15E-4
		20×20	1.79E-5	3.94E-6
		25×25	2.56E-6	3.72E-7
		30×30	6.68E-8	8.31E-9

Table 11 Appendix C (variable-coefficient Poisson equation): Comparison of the maximum and rms errors versus the number of collocation points (Q) obtained by HLConcELM and conventional ELM. In all cases, the neural network has 3 hidden layers and a total of 900 trainable parameters for both HLConcELM and conventional ELM. For HLConcELM, the hidden magnitude vector $\mathbf{R} = (3.0, 0.005, 0.15)$ for the network $[2, 800, 50, 50, 1]$ and $\mathbf{R} = (0.5, 0.5, 0.6)$ for the network $[2, 50, 50, 800, 1]$. For conventional ELM, the hidden-layer coefficients are assigned to uniform random values from $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m obtained using the method of [13], with $R_{m0} = 0.4$ for the network $[2, 800, 50, 900, 1]$ and $R_{m0} = 0.5$ for the network $[2, 50, 50, 900, 1]$.

Table 10 are attained with two hidden layers in the neural network and a total of 850 training parameters. The results in Table 11 correspond to three hidden layers in the neural network with a total of 900 training parameters. The HLConcELM data in Table 10 for the networks $[2, 800, 50, 1]$ and $[2, 50, 800, 1]$ correspond to those in Table 1. The simulation parameter values are listed in the tables or provided in the table captions. The exponential convergence of the errors with respect to the number of collocation points is evident in all test cases. The error levels from HLConcELM and the conventional ELM are close, reaching the order around 10^{-8} in terms of the maximum error and 10^{-9} in terms of the rms error. The error values resulting from HLConcELM in general appear better than those from the Conventional ELM, e.g. by comparing the HLConcELM results (with $[2, 800, 50, 1]$) and the conventional ELM results (with $[2, 800, 850, 1]$) in Table 10 or comparing the HLConcELM results (with $[2, 800, 50, 50, 1]$) and the conventional ELM results (with $[2, 800, 50, 900, 1]$) in Table 11. But this is not true for every test case; see e.g. the case Q=25×25 between HLConcELM (with $[2, 50, 800, 1]$) and conventional ELM (with $[2, 50, 850, 1]$) in Table 10 or the cases Q=15×15 and 20×20 between HLConcELM (with $[2, 50, 50, 800, 1]$) and conventional ELM (with $[2, 50, 50, 900, 1]$) in Table 11.

method	network	Q	max error	rms error
HLConcELM	$[2, 500, 30, 1]$	5×5	4.00E+0	1.48E+0
		10×10	1.59E+0	2.80E-1
		15×15	1.27E-3	1.62E-4
		20×20	1.27E-5	2.34E-6
		25×25	2.08E-6	2.11E-7
		30×30	3.74E-6	3.48E-7
Conventional ELM	$[2, 500, 530, 1]$	5×5	3.75E+0	8.94E-1
		10×10	3.46E-1	4.03E-2
		15×15	1.05E-3	1.92E-4
		20×20	8.63E-5	1.44E-5
		25×25	3.90E-5	3.07E-6
		30×30	3.50E-5	3.84E-6
HLConcELM	$[2, 30, 500, 1]$	5×5	3.23E+0	8.43E-1
		10×10	7.22E-1	1.32E-1
		15×15	1.06E-3	2.36E-4
		20×20	2.56E-5	3.12E-6
		25×25	8.78E-7	1.38E-7
		30×30	8.99E-7	8.20E-8
Conventional ELM	$[2, 30, 530, 1]$	5×5	3.51E+0	9.04E-1
		10×10	6.57E-1	1.11E-1
		15×15	3.87E-4	8.38E-5
		20×20	1.67E-5	2.07E-6
		25×25	2.74E-6	3.22E-7
		30×30	1.87E-6	1.73E-7

Table 12 Appendix C (nonlinear Helmholtz equation): Comparison of the maximum and rms errors versus the number of collocation points (Q) obtained by HLConcELM and conventional ELM. In all cases, the neural network has two hidden layers and a total of 530 trainable parameters for both HLConcELM and conventional ELM. The HLConcELM data in this table correspond to those in Table 4. For conventional ELM, the hidden-layer coefficients are assigned to uniform random values from $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m obtained using the method of [13], with $R_{m0} = 0.4$ for the network $[2, 500, 530, 1]$ and $R_{m0} = 0.6$ for the network $[2, 30, 530, 1]$.

Tables 12 and 13 show the comparisons between HLConcELM and conventional ELM for the nonlinear Helmholtz problem from Section 3.3. The results in Table 12 correspond to two hidden layers in the neural network with a total of 530 training parameters, and those in Table 13 correspond to three hidden layers in the neural network with a total of 560 training parameters. The simulation parameter values are provided in the table captions or listed in the tables. Note that the HLConcELM data in Table 12 correspond to those in Table 4 with the networks $[2, 500, 30, 1]$ and $[2, 30, 500, 1]$. The relative performance between HLConcELM and conventional ELM exhibited by these data is similar to what has been observed from Tables 10 and 11 for the variable-coefficient Poisson equation. The error levels resulting from HLConcELM and conventional ELM are quite close, on the order of 10^{-6} or 10^{-7} in terms of the maximum error and 10^{-7} or 10^{-8} in terms of the rms error. Overall the error values from HLConcELM appear slightly better than those from the conventional ELM; see e.g. those data in Table 12 and the cases between HLConcELM with $[2, 30, 30, 500, 1]$ and conventional ELM with $[2, 30, 30, 560, 1]$ in Table 13. But this is not consistently so for all the test cases; see e.g. the cases between HLConcELM with $[2, 500, 30, 30, 1]$ and conventional ELM with $[2, 500, 30, 560, 1]$ in Table 13.

method	network	Q	max error	rms error
HLConcELM	$[2, 500, 30, 30, 1]$	5×5	5.74E+0	1.85E+0
		10×10	6.84E-1	1.98E-1
		15×15	4.45E-3	9.10E-4
		20×20	1.78E-5	3.46E-6
		25×25	1.85E-6	1.44E-7
		30×30	1.52E-6	1.86E-7
Conventional ELM	$[2, 500, 30, 560, 1]$	5×5	2.89E+0	9.17E-1
		10×10	6.18E-1	1.06E-1
		15×15	2.05E-3	3.99E-4
		20×20	3.88E-5	5.72E-6
		25×25	9.46E-7	1.17E-7
		30×30	3.51E-7	4.24E-8
HLConcELM	$[2, 30, 30, 500, 1]$	5×5	3.30E+0	8.69E-1
		10×10	2.17E-1	4.33E-2
		15×15	4.42E-3	7.98E-4
		20×20	7.43E-5	1.53E-5
		25×25	6.38E-6	1.06E-6
		30×30	5.82E-6	3.73E-7
Conventional ELM	$[2, 30, 30, 560, 1]$	5×5	4.51E+0	1.04E+0
		10×10	1.47E+0	1.05E-1
		15×15	8.24E-1	1.53E-1
		20×20	2.36E-2	2.90E-3
		25×25	4.49E-5	7.54E-6
		30×30	6.33E-6	6.68E-7

Table 13 Appendix C (nonlinear Helmholtz equation): Comparison of the maximum and rms errors versus the number of collocation points (Q) obtained by HLConcELM and conventional ELM. In all cases, the neural network has 3 hidden layers and a total of 560 trainable parameters for both HLConcELM and conventional ELM. For HLConcELM, the hidden magnitude vector $\mathbf{R} = (2.1, 0.2, 2.3)$ for the network $[2, 500, 30, 30, 1]$ and $\mathbf{R} = (0.82, 0.26, 0.34)$ for the network $[2, 30, 30, 500, 1]$. For conventional ELM, the hidden-layer coefficients are assigned to uniform random values from $[-R_m, R_m]$ with $R_m = R_{m0}$. Here R_{m0} is the optimal R_m obtained using the method of [13], with $R_{m0} = 0.37$ for the network $[2, 500, 30, 560, 1]$ and $R_{m0} = 0.55$ for the network $[2, 30, 30, 560, 1]$.

It is noted that in all these test cases the neural network for the conventional ELM has a wide last hidden layer. This is consistent with the observation that the conventional ELM is only accurate when the last hidden layer is wide.

Appendix D. Laplace Equation Around a Reentrant Corner

This appendix provides a test of the HLConcELM method with the Laplace equation around a reentrant corner, where the solution is not smooth. Figure 25 is a sketch of the L-shaped domain $\Omega = \overline{OABCDEO}$ (with an reentrant corner at O) employed in this test. We consider the following problem on Ω,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad (x, y) \in \Omega, \tag{37a}$$

$$u(x, y) = 0, \quad (x, y) \in \overline{OA},\ \overline{OE}, \tag{37b}$$

$$u(x, y) = r^{\frac{2k}{3}} \sin\left(\frac{2k}{3}\theta\right), \quad (x, y) \in \overline{AB},\ \overline{BC},\ \overline{CD},\ \overline{DE}, \tag{37c}$$

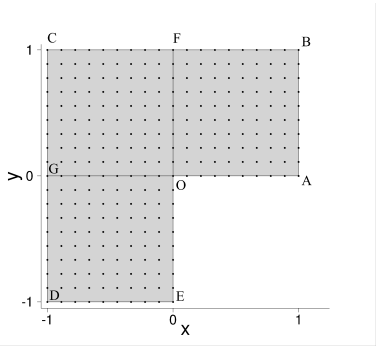

Fig. 25 Appendix D (reentrant corner): sketch of the L-shaped domain $\overline{OABCDEO}$ with an reentrant corner at O. The sketch shows an example set of $Q = 3\times(10\times10)$ uniform collocation points, with 10×10 points in each of the three regions $\overline{OABF}$, $\overline{OFCG}$ and $\overline{OGDE}$.

	k=1		k=2		k=3		k=5	
M	max-err	rms-err	max-err	rms-err	max-err	rms-err	max-err	rms-err
100	3.47E-1	6.81E-2	1.97E-1	5.52E-2	3.22E-1	8.56E-2	1.05E+0	3.01E-1
200	2.12E-1	2.59E-2	8.03E-2	1.47E-2	2.18E-2	3.28E-3	8.91E-2	1.45E-2
300	1.71E-1	1.64E-2	6.04E-2	8.43E-3	4.33E-4	5.47E-5	3.47E-3	6.65E-4
400	1.53E-1	1.28E-2	4.04E-2	4.90E-3	5.76E-6	7.44E-7	2.56E-3	3.60E-4
500	1.37E-1	1.01E-2	3.58E-2	3.82E-3	2.01E-7	2.84E-8	1.68E-3	1.91E-4
600	1.21E-1	7.75E-3	2.65E-2	2.46E-3	7.72E-9	7.24E-10	9.08E-4	9.02E-5
700	1.11E-1	6.46E-3	2.28E-2	1.87E-3	4.69E-10	3.92E-11	6.19E-4	5.48E-5
800	1.02E-1	5.47E-3	1.98E-2	1.50E-3	1.65E-11	1.06E-12	4.27E-4	3.52E-5
900	9.62E-2	4.85E-3	1.78E-2	1.32E-3	1.80E-11	2.60E-12	3.33E-4	2.64E-5
1000	8.94E-2	6.96E-3	1.57E-2	1.23E-3	1.37E-11	2.55E-12	3.00E-4	3.14E-5

Table 14 Appendix D (reentrant corner): The maximum/rms errors of the HLConcELM solution versus the number of nodes in the first hidden layer (M) for solution fields with different regularity (k parameter). In HLConcELM, neural network $[2, M, 50, 1]$, Gaussian activation function, $Q = 3 \times (20 \times 20)$ uniform collocation points, $\mathbf{R} = (5.0, 0.1)$. The solution field is smooth if k is a multiple of 3, and non-smooth otherwise. The (non-smooth) solution is smoother if k is larger.

where $u(x, y)$ is the field to be solved for, (r, θ) denotes the polar coordinate, and $k \geqslant 1$ is a prescribed integer. This problem has the following solution,

$$u(x,y) = r^{\frac{2k}{3}} \sin\left(\frac{2k}{3}\theta\right), \quad (x,y) \in \Omega. \tag{38}$$

The integer k influences the regularity of the solution. If k is a multiple of 3, then the solution $u(x, y)$ is smooth (C^∞) on Ω. Otherwise, the solution is non-smooth, with its $\lceil \frac{2k}{3} \rceil$-th derivative being singular at the reentrant corner. We solve this problem by the HLConcELM method, and employ a set of uniform grid points in the sub-regions $\overline{OABF}$, $\overline{OFCG}$ and $\overline{OGDE}$ as the collocation points. Figure 25 shows a set of $Q = 3 \times (10 \times 10)$ uniform collocation points on the domain as an example. The Gaussian activation function is employed in the neural network. We employ a fixed seed value 10 for the random number generators.

Figure 26 shows distributions of the exact solution (38), the HLConcELM solution and its point-wise absolute error, corresponding to three different solution fields with $k = 1$, 3 and 5. The values for the simulation parameters are provided

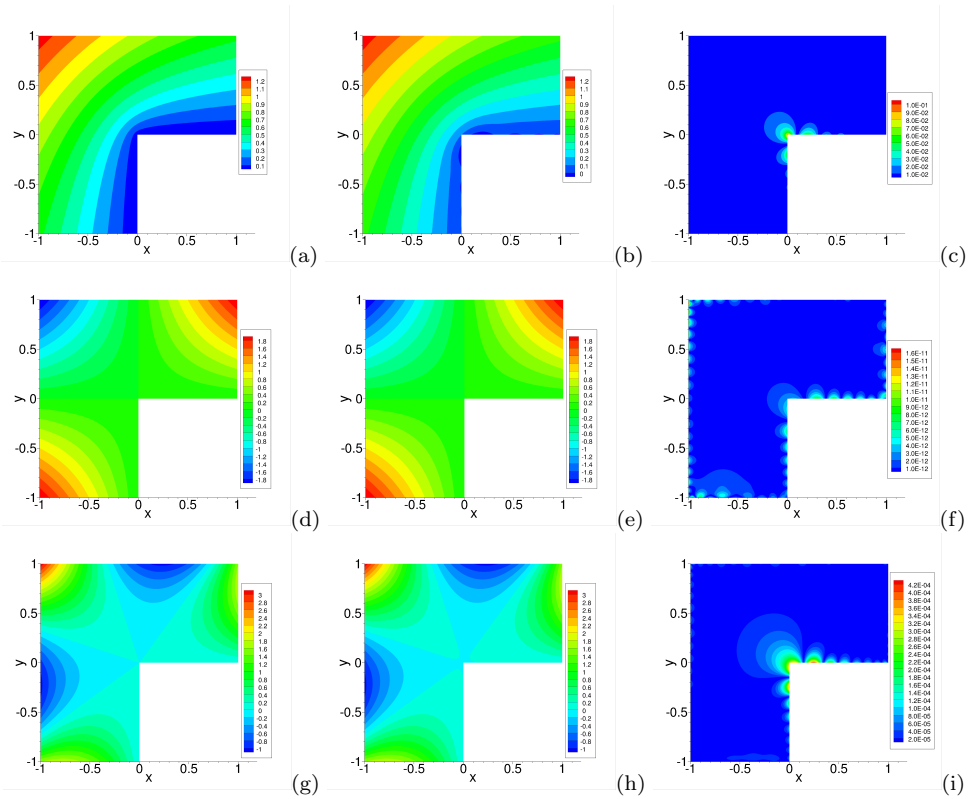

Fig. 26 Appendix D (reentrant corner): Distributions of the exact solution (a,d,g), the HLConcELM solution (b,e,h), and the point-wise absolute error of the HLConcELM solution (c,f,i) to the Laplace equation. (a,b,c) $k = 1$ (non-smooth), (d,e,f) $k = 3$ (smooth), and (g,h,i) $k = 5$ (non-smooth) in the solution field. In HLConcELM, neural network $[2, 800, 50, 1]$, Gaussian activation function, $Q = 3 \times (20 \times 20)$ uniform collocation points, $\mathbf{R} = (5.0, 0.1)$.

	k=1		k=2		k=3		k=5	
Q	max-err	rms-err	max-err	rms-err	max-err	rms-err	max-err	rms-err
3×(5×5)	1.75E-1	6.15E-2	9.30E-2	2.60E-2	9.47E-2	3.15E-2	2.33E-1	8.27E-2
3×(10×10)	4.65E-1	6.92E-2	7.92E-2	1.25E-2	5.09E-4	9.55E-5	1.59E-3	2.31E-4
3×(20×20)	1.02E-1	5.47E-3	1.98E-2	1.50E-3	1.65E-11	1.06E-12	4.27E-4	3.52E-5
3×(30×30)	1.14E-1	6.81E-3	2.33E-2	1.93E-3	1.70E-11	1.51E-12	6.40E-4	5.75E-5

Table 15 Appendix D (reentrant corner): The maximum/rms errors of the HLConcELM solution versus the number of collocation points (Q) for solution fields with different regularity. In HLConcELM, neural network $[2, 800, 50, 1]$, Gaussian activation function, $\mathbf{R} = (5.0, 0.1)$.

in the figure caption. The HLConcELM result is extremely accurate for the case with a smooth solution ($k = 3$), with the maximum error on the order 10^{-11} in the domain. On the other hand, the HLConcELM solution is much less accurate for the non-smooth cases ($k = 1, 5$), with the maximum error around 10^{-1} for $k = 1$ and around 10^{-4} for $k = 5$. One can note that the computed HLConcELM solution is more accurate for a smoother solution field (larger k).

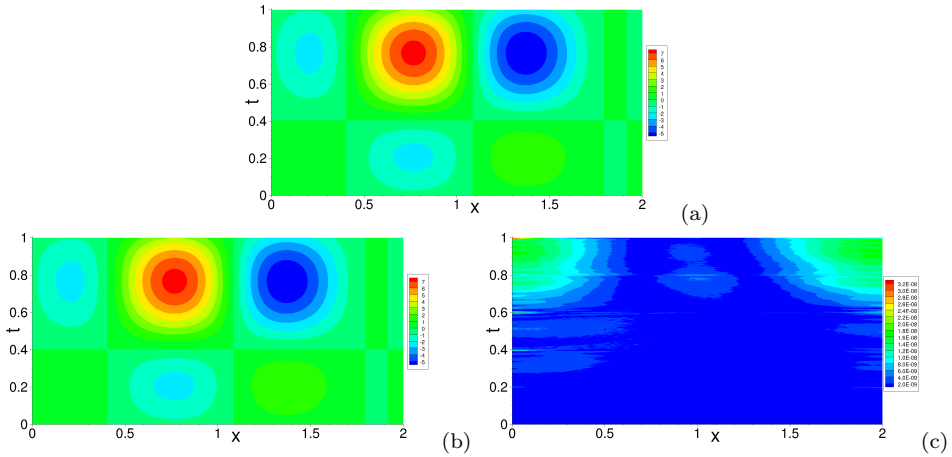

Fig. 27 (Appendix E) Kuramoto-Sivashinsky equation (case #1): Distributions of (a) the exact solution, (b) the HLConcELM solution and (c) its point-wise absolute error. In HLConcELM, 5 uniform time blocks, neural network $\mathbf{M} = [2, 400, 50, 1]$ and $Q = 25 \times 25$ uniform collocation points per time block, hidden magnitude vector $\mathbf{R} = (1.64, 0.05)$, Gaussian activation function.

Tables 14 and 15 illustrate the convergence behavior of the HLConcELM errors with respect to the number of hidden nodes in the neural network and the number of collocation points (Q). Several cases corresponding to smooth and non-smooth solution fields are shown. The simulation parameter values are provided in the captions of these tables. The neural network architecture is given by $[2, M, 50, 1]$, where M is either fixed at $M = 800$ or varied systematically. The set of collocation points is either fixed at $Q = 3 \times (20 \times 20)$ or varied systematically. For the smooth case ($k = 3$), the HLConcELM solution exhibits an exponential convergence with respect to M and Q. For the non-smooth cases ($k = 1, 2, 5$), the convergence is much slower and in general quite slow. However, if the solution is smoother (larger k), we can generally observe an initial exponential decrease in the HLConcELM errors as M or Q increases, and that the error reduction slows down as M or Q reaches a certain level. For example, with the case $k = 5$ one can observe in Table 14 the initial exponential decrease in the errors with increasing M for $M \leqslant 300$.

Appendix E. Kuramoto-Sivashinsky Equation

This appendix provides a test of the HLConcELM method with the Kuramoto-Sivashinsky equation [38,55]. We consider the domain $(x, t) \in \Omega = [a, b] \times [0, t_f]$, and the Kuramoto-Sivashinsky equation on Ω with periodic boundary conditions,

neural network	collocation points	max error	rms error
$[2, 400, 50, 1]$	5×5	2.45E+1	5.80E+0
	10×10	2.18E-1	4.86E-2
	15×15	2.62E-4	6.95E-5
	20×20	2.55E-7	8.08E-8
	25×25	3.44E-8	4.57E-9
	30×30	1.61E-8	7.52E-10
$[2, 50, 400, 1]$	5×5	1.96E+1	5.26E+0
	10×10	4.48E+0	9.88E-1
	15×15	2.59E-4	4.77E-5
	20×20	2.57E-6	5.08E-7
	25×25	4.32E-7	8.16E-8
	30×30	7.40E-8	2.91E-9

Table 16 (Appendix E) Kuramoto-Sivashinsky equation (case #1): The maximum/rms errors of the HLConcELM solution versus the number of collocation points (Q) on two neural networks. 5 uniform time blocks, hidden magnitude vector $\mathbf{R} = (1.64, 0.05)$ for the network $[2, 400, 50, 1]$ and $\mathbf{R} = (0.64, 0.2)$ for the network $[2, 50, 400, 1]$.

neural network	M	max error	rms error
$[2, M, 50, 1]$	100	2.64E+1	6.41E+0
	200	1.84E-3	3.85E-4
	300	9.00E-7	1.87E-7
	400	3.44E-8	4.57E-9
	500	1.05E-8	1.32E-9
$[2, 50, M, 1]$	100	3.20E+1	9.54E+0
	200	8.76E-4	1.91E-4
	300	6.53E-7	8.31E-8
	400	4.32E-7	8.16E-8
	500	2.05E-7	2.30E-8

Table 17 (Appendix E) Kuramoto-Sivashinsky equation (case #1): The maximum/rms errors of the HLConcELM solution versus the number of nodes (M) on two network architectures $[2, M, 50, 1]$ and $[2, 50, M, 1]$ (M varied). 5 uniform time blocks, $Q = 25 \times 25$ uniform collocation points per time block, hidden magnitude vector $\mathbf{R} = (1.64, 0.05)$ for the architecture $[2, M, 50, 1]$ and $\mathbf{R} = (0.64, 0.2)$ for the architecture $[2, 50, M, 1]$.

$$\frac{\partial u}{\partial t} + \alpha u \frac{\partial u}{\partial x} + \beta \frac{\partial^2 u}{\partial x^2} + \gamma \frac{\partial^4 u}{\partial x^4} = f(x,t), \tag{39a}$$

$$u(a,t) = u(b,t), \quad \left.\frac{\partial u}{\partial x}\right|_{(a,t)} = \left.\frac{\partial u}{\partial x}\right|_{(b,t)}, \tag{39b}$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{(a,t)} = \left.\frac{\partial^2 u}{\partial x^2}\right|_{(b,t)}, \quad \left.\frac{\partial^3 u}{\partial x^3}\right|_{(a,t)} = \left.\frac{\partial^3 u}{\partial x^3}\right|_{(b,t)}, \tag{39c}$$

$$u(x,0) = g(x). \tag{39d}$$

In these equations, (α, β, γ) are constants, $u(x,t)$ is the field function to be solved for, f is a prescribed source term, and g denotes the initial distribution. The domain parameters a, b and t_f will be specified below. We solve this problem by the locHLConcELM method (see Remark 6) together with the block time marching scheme (see Remark 5). The seed for the random number generators is set to 100 in the following tests.

Case #1: Manufactured Analytic Solution. We first consider a manufactured analytic solution to (39) to illustrate the convergence behavior of HLConcELM. We employ

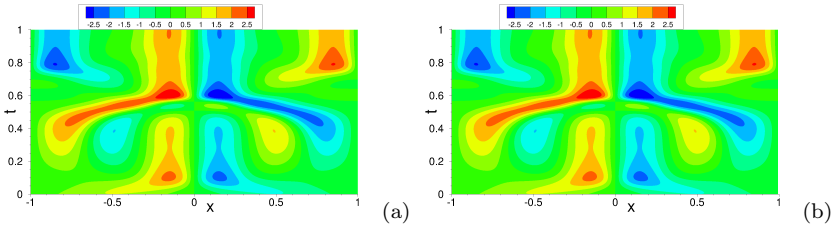

Fig. 28 (Appendix E) Kuramoto-Sivashinsky equation (case #2): Distributions of (a) the locHLConcELM solution and (b) the Chebfun solution. In Chebfun, 400 Fourier grid points in x, time step size $\Delta t = 1e-4$. In locHLConcELM, 20 uniform time blocks, 4 uniform sub-domains along x within each time block, neural network $\mathbf{M} = [2, 400, 1]$ and $Q = 25 \times 25$ uniform collocation points on each sub-domain, hidden magnitude vector $\mathbf{R} = 8.0$, sine activation function.

the following parameter values,

$$a = 0, \quad b = 2, \quad t_f = 1, \quad \alpha = 1, \quad \beta = 1, \quad \gamma = 0.1,$$

and the analytic solution given by

$$u(x,t) = \left[\frac{3}{2}\cos\left(\pi x + \frac{7\pi}{20}\right) + \frac{27}{20}\cos\left(2\pi x - \frac{3\pi}{5}\right)\right]\left[\frac{3}{2}\cos\left(\pi t + \frac{7\pi}{20}\right) + \frac{27}{20}\cos\left(2\pi t - \frac{3\pi}{5}\right)\right]. \tag{40}$$

The source term f and the initial distribution g are chosen such that the expression (40) satisfies the system (39). The distribution of this solution is shown in Figure 27(a).

The distributions of the HLConcELM solution and its point-wise absolute error are shown in Figures 27(b) and (c). We have employed 5 uniform time blocks in the HLConcELM simulation, and a neural network architecture $[2, 400, 50, 1]$ with the Gaussian activation function within each time block. The other simulation parameter values are provided in the caption of Figure 27. The HLConcELM method captures the solution accurately, with the maximum error on the order 10^{-8} in the spatial-temporal domain.

Tables 16 and 17 illustrate the exponential convergence behavior of the HLConcELM accuracy with respect to the collocation points and the network size for the Kuramoto-Sivashinsky equation. Table 16 lists the HLConcELM errors versus the number of collocation points (Q) obtained with two neural networks, with a narrow and wide last hidden layer, respectively. Table 17 shows the HLConcELM errors versus the number of nodes (M) in the first or the last hidden layer of the neural network, obtained with a fixed set of $Q = 25 \times 25$ uniform collocation points. The captions of these tables provide the parameter values in these simulations. It can be observed that the HLConcELM errors decrease exponentially as the number of collocation points or the network size increases.

Case #2: No Exact Solution and Comparison with Chebfun. We next consider the following parameter values and settings:

$$a = -1, \quad b = 1, \quad t_f = 1, \quad \alpha = 5, \quad \beta = 0.5, \quad \gamma = 0.005,$$
$$f(x,t) = 0, \quad g(x) = -\sin(\pi x).$$

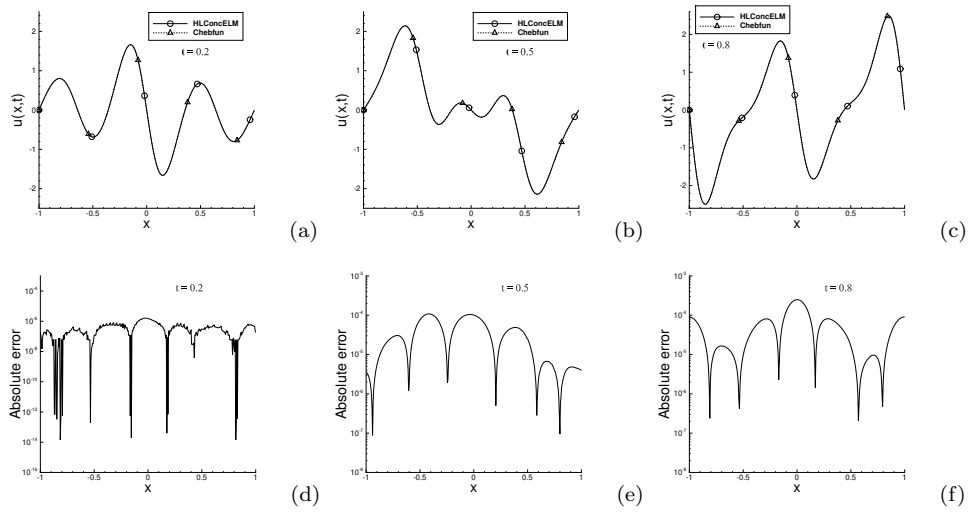

Fig. 29 (Appendix E) Kuramoto-Sivashinsky equation (case #2): Comparison of solution profiles between locHLConcELM and Chebfun at $t = 0.2$ (a), $t = 0.5$ (b), and $t = 0.8$ (c). Profiles of the absolution error between the Chebfun and the HLConcELM solutions at $t = 0.2$ (d), $t = 0.5$ (e), and $t = 0.8$ (f). Simulation settings and parameters follow those of Figure 28.

The exact solution for this case is unknown. We will employ the result computed by the software package Chebfun [14], with a sufficient resolution, as the reference solution to compare with HLConcELM.

Figure 28 shows the solution distributions obtained by the locHLConcELM method and by Chebfun in the spatial-temporal domain for this case. With locHLConcELM, we have employed 20 uniform time blocks, 4 uniform sub-domains (along the x direction) within each time block, and a local neural network $[2, 400, 1]$ with $Q = 25 \times 25$ uniform collocation points on each sub-domain. The sine activation, $\sigma(x) = \sin(x)$, has been employed with the local neural networks. The Chebfun solution is obtained with 400 Fourier grad points along the x direction and a time step size $\Delta t = 10^{-4}$. The locHLConcELM solution agrees well with the Chebfun solution qualitatively.

Figure 29 provides quantitative comparisons between locHLConcELM and Chebfun for this case. It compares the solution profiles obtained by these two methods at three time instants $t = 0.2$, 0.5 and 0.8 (top row), and also shows the corresponding profiles of the absolute error between these two methods (bottom row). No difference can be discerned from the solution profiles between locHLConcELM and Chebfun. The errors between these two methods generally increase over time, with the maximum error on the order 10^{-6} at $t = 0.2$ and 10^{-4} at $t = 0.5$ and 0.8. These results indicate that the current method has captured the solution quite accurately.

Case #3: Another Comparison With Chebfun. We consider still another set of problem parameters as follows:

$$a = -1, \quad b = 1, \quad t_f = 0.25, \quad \alpha = 6, \quad \beta = 0.5, \quad \gamma = 0.001,$$
$$f(x,t) = 0, \quad g(x) = -\sin(\pi x).$$

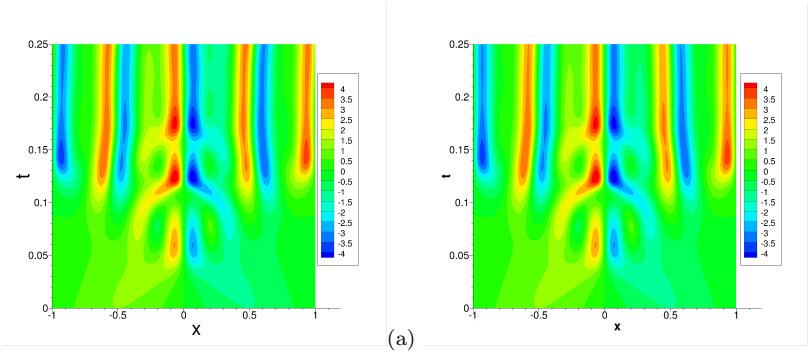

Fig. 30 (Appendix E) Kuramoto-Sivashinsky equation (case #3): Distributions of (a) the locHLConcELM solution and (b) the Chebfun solution. In Chebfun, 1000 Fourier grid points in x, time step size $\Delta t = 1e-5$. In locHLConcELM, 12 time blocks (time block size: 0.025 for the first 8 time blocks, 0.0125 for the last 4 time blocks), 10 uniform sub-domains along x within each time block, neural network $\mathbf{M} = [2, 300, 1]$ and $Q = 21 \times 21$ uniform collocation points within each sub-domain, hidden magnitude vector $\mathbf{R} = 2.5$, sinc activation function.

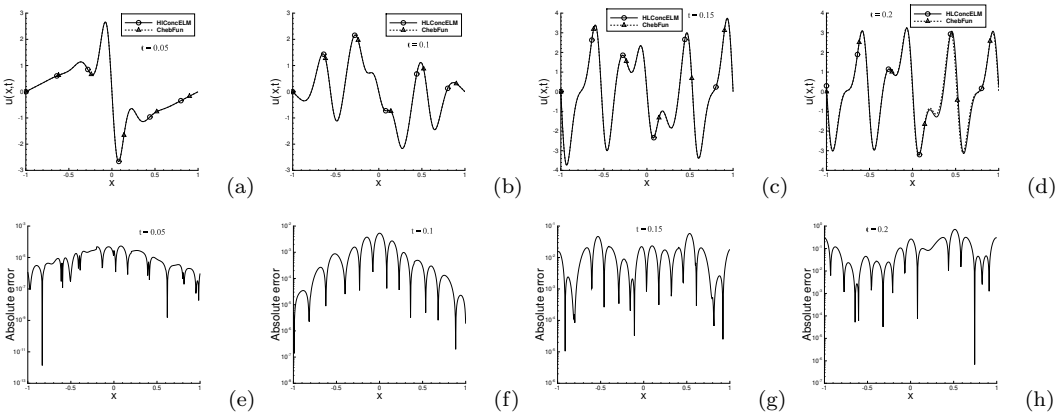

Fig. 31 (Appendix E) Kuramoto-Sivashinsky equation (case #3): Comparison of solution profiles between locHLConcELM and Chebfun at (a) $t = 0.05$, (b) $t = 0.1$, (c) $t = 0.15$, and (d) $t = 0.2$. Profiles of the absolute error between the locHLConcELM solution and the Chebfun solution at (e) $t = 0.05$, (f) $t = 0.1$, (g) $t = 0.15$, and (h) $t = 0.2$. Simulation settings and parameters follow those of Figure 30.

We again compare the HLConcELM result with the reference solution computed by Chebfun.

Figure 30 compares distributions of the locHLConcELM solution and the Chebfun solution. With locHLConcELM, we have employed 12 time blocks, 10 uniform sub-domains along the x direction within each time block, a local neural network $[2, 300, 1]$ and a uniform set of $Q = 21 \times 21$ collocation points on each sub-domain. The random magnitude vector is $\mathbf{R} = 2.5$, and the sinc activation function ($\sigma(x) = \frac{\sin(\pi x)}{\pi x}$) is employed. With Chebfun, we have employed 1000 Fourier grid points along the x direction and a time step size $\Delta t = 10^{-5}$. The distribution of the locHLConcELM solution is qualitatively similar to that of the Chebfun solution.

Figure 31 provides a quantitative comparison of the solution profiles between locHLConcELM and Chebfun at several time instants (top row), and also shows the corresponding profiles of the absolute error between the locHLConcELM solution and the Chebfun solution (bottom row). The locHLConcELM solution agrees very well with the Chebfun solution initially, and the difference between these two solutions grows over time.

Appendix F. Schrodinger Equation

This appendix provides a test of the HLConcELM method with the Schrodinger equation. We consider the domain $(x,t) \in \Omega = [a,b] \times [0,t_f]$, and the Schrodinger equation on Ω with periodic boundary conditions:

$$i\frac{\partial h}{\partial t} + \frac{1}{2}\frac{\partial^2 h}{\partial x^2} + |h|^2 h = f(x,t), \tag{41a}$$

$$h(a,t) = h(b,t), \quad \left.\frac{\partial h}{\partial x}\right|_{(a,t)} = \left.\frac{\partial h}{\partial x}\right|_{(b,t)}, \tag{41b}$$

$$h(x,0) = g(x), \tag{41c}$$

where $h(x,t)$ is the complex field function to be solved for, $f(x,t)$ is a prescribed complex source term, and $g(x)$ is the initial distribution. Let $h = u(x,t) + iv(x,t)$, where u and v denote the real and the imaginary parts of h, respectively. The domain parameters a, b and t_f will be specified below.

We solve this problem by the HLConcELM method, or the locHLConcELM method (see Remark 6), combined with the block time marching scheme (see Remark 5). The input layer of the neural network consists of two nodes, representing x and t, respectively. The output layer consists also of two nodes, representing the real part and the imaginary part of $h(x,t)$, respectively. Accordingly, the system (41) is re-written into an equivalent form in terms of the real part and the imaginary part of $h(x,t)$. The reformulated system is employed in the HLConcELM simulation. When multiple sub-domains are employed in locHLConcELM, we impose C^1 continuity conditions along the x direction and C^0 continuity conditions along the t direction across the shared sub-domain boundaries. The seed for the random number generators is set to 100 in the HLConcELM simulations.

Case #1: Manufactured Analytic Solution. We first illustrate the convergence behavior of HLConcELM using a manufactured analytic solution. We employ the following domain parameters,

$$a = -1, \quad b = 1, \quad t_f = 1.5,$$

and the analytic solution $h = u + iv$, where

$$\begin{cases} u(x,t) = \left[\frac{3}{2}\sin\left(\pi x + \frac{7\pi}{20}\right) + \frac{27}{20}\sin\left(2\pi x - \frac{3\pi}{5}\right)\right]\left[\frac{3}{2}\sin\left(\pi t + \frac{7\pi}{20}\right)\right. \\ \qquad \left. + \frac{27}{20}\sin\left(2\pi t - \frac{3\pi}{5}\right)\right], \\ v(x,t) = \left[\frac{5}{4}\cos\left(\pi x + \frac{7\pi}{20}\right) + \frac{3}{2}\cos\left(2\pi x - \frac{3\pi}{5}\right)\right]\left[\frac{5}{4}\cos\left(\pi t + \frac{7\pi}{20}\right)\right. \\ \qquad \left. + \frac{3}{2}\cos\left(2\pi t - \frac{3\pi}{5}\right)\right]. \end{cases} \tag{42}$$

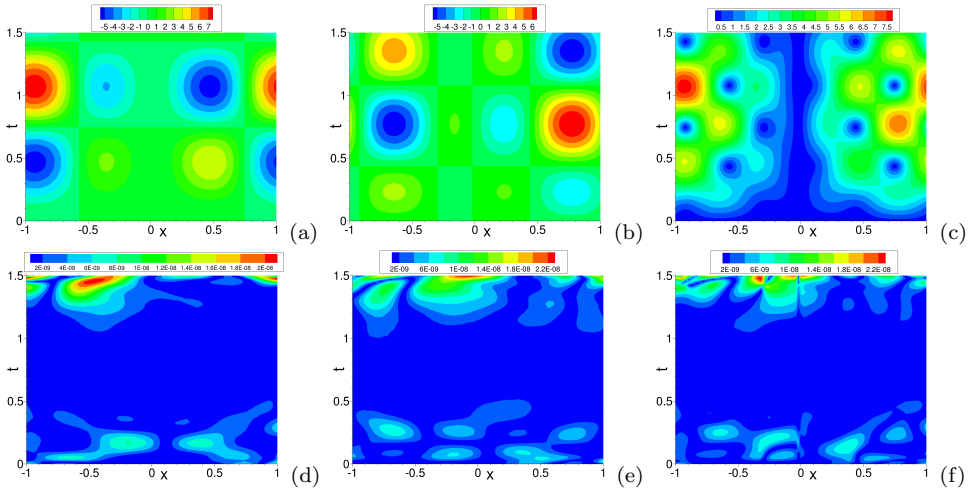

Fig. 32 (Appendix F) Schrodinger equation (case #1): Distributions of the real part (a) and its point-wise absolute error (d), the imaginary part (b) and its point-wise absolute error (e), the norm (c) and its point-wise absolute error (f), of the HLConcELM solution $h(x,t)$. Neural network $[2, 400, 30, 2]$, Gaussian activation function, $Q = 25 \times 25$ uniform collocation points, hidden magnitude vector $\mathbf{R} = (1.7, 0.01)$.

neural network	Q	real(h)		imag(h)		$\|h\|$	
		max-err	rms-err	max-err	rms-err	max-err	rms-err
[2,400,30,1]	5×5	1.04E+1	3.11E+0	9.32E+0	2.86E+0	6.43E+0	1.81E+0
	10×10	6.44E+0	9.78E-1	1.01E+1	1.63E+0	9.27E+0	1.43E+0
	15×15	3.66E-5	3.79E-6	6.32E-5	5.79E-6	5.68E-5	5.29E-6
	20×20	9.32E-8	9.93E-9	1.09E-7	1.03E-8	1.23E-7	1.01E-8
	25×25	2.23E-8	3.39E-9	2.32E-8	3.47E-9	2.25E-8	3.20E-9
	30×30	1.51E-8	1.94E-9	1.52E-8	2.32E-9	1.52E-8	2.13E-9
[2,30,400,1]	5×5	8.51E+0	2.46E+0	1.85E+1	4.74E+0	1.27E+1	3.18E+0
	10×10	3.12E-1	3.20E-2	1.32E-1	3.63E-2	2.43E-1	3.23E-2
	15×15	5.37E-4	4.45E-5	6.30E-4	4.71E-5	7.16E-4	4.72E-5
	20×20	3.86E-6	2.95E-7	1.93E-6	2.18E-7	3.77E-6	2.32E-7
	25×25	2.75E-7	3.31E-8	2.75E-7	3.23E-8	2.74E-7	3.03E-8
	30×30	1.55E-7	3.16E-8	1.90E-7	3.65E-8	1.95E-7	3.14E-8

Table 18 (Appendix F) Schrodinger equation (case #1): The maximum/rms errors of the HLConcELM solution versus the number of collocation points (Q) on two neural networks. Hidden magnitude vector $\mathbf{R} = (1.7, 0.01)$ for the network $[2, 400, 30, 1]$ and $\mathbf{R} = (0.86, 0.25)$ for the network $[2, 30, 400, 1]$. "real(h)", "imag(h)" and $|h|$ refer to the real part, the imaginary part, and the norm of $h(x,t)$.

The source term $f(x,t)$ and the initial distribution $g(x,t)$ are chosen such that the expression (42) satisfies the system (41).

Figure 32 shows distributions of the real part u, the imaginary part v, and the norm $|h|$ of the HLConcELM solution $h(x,t)$, as well as their point-wise absolute errors when compared with the analytic solution (42), in the spatial-temporal domain. The neural network architecture is given by $\mathbf{M} = [2, 400, 30, 2]$, and the other simulation parameter values are listed in the figure caption. The HLConcELM solution is observed to be highly accurate, and the maximum error on the order 10^{-8} for all of these quantities.

neural network	M	real(h)		imag(h)		$\|h\|$	
		max-err	rms-err	max-err	rms-err	max-err	rms-err
[2,M,30,1]	50	1.13E+0	1.59E-1	2.84E+0	3.00E-1	1.85E+0	2.44E-1
	100	1.21E-1	1.91E-2	1.04E-1	2.04E-2	1.14E-1	1.95E-2
	200	9.07E-5	1.45E-5	1.05E-4	1.64E-5	9.55E-5	1.53E-5
	300	1.70E-7	3.38E-8	1.98E-7	3.62E-8	1.99E-7	3.34E-8
	400	2.23E-8	3.39E-9	2.32E-8	3.47E-9	2.25E-8	3.20E-9
	500	1.42E-8	1.20E-9	1.79E-8	1.37E-9	1.55E-8	1.31E-9
[2,30,M,1]	50	3.36E+0	3.70E-1	2.43E+0	3.39E-1	3.60E+0	3.75E-1
	100	9.70E-2	1.08E-2	7.37E-2	1.54E-2	9.52E-2	1.44E-2
	200	1.57E-3	1.53E-4	1.46E-3	1.51E-4	2.04E-3	1.52E-4
	300	1.21E-5	2.09E-6	1.78E-5	1.96E-6	1.36E-5	1.93E-6
	400	2.75E-7	3.31E-8	2.75E-7	3.23E-8	2.74E-7	3.03E-8
	500	1.70E-7	2.35E-8	9.68E-8	1.91E-8	1.57E-7	2.13E-8

Table 19 (Appendix F) Schrodinger equation (case #1): The maximum/rms errors of the HLConcELM solution versus the number of nodes (M) on two network architectures $[2, M, 30, 2]$ and $[2, 30, M, 2]$ (M varied). $Q = 25 \times 25$ uniform collocation points, hidden magnitude vector $\mathbf{R} = (1.7, 0.01)$ for the architecture $[2, M, 30, 1]$ and $\mathbf{R} = (0.86, 0.25)$ for the architecture $[2, 30, M, 1]$.

The exponential convergence of the HLConcELM accuracy is illustrated by the data in Tables 18 and 19. Table 18 lists the maximum and rms errors of the real part, the imaginary part, and the norm of $h(x,t)$ as a function of the number of collocation points (Q) obtained by HLConcELM on two neural networks with a narrow and a wide last hidden layer, respectively. Table 19 lists the HLConcELM errors for the real/imaginary parts and the norm of $h(x,t)$ on two network architectures having two hidden layers, with the number of nodes (M) in the first or the last hidden layer varied. The values for the simulation parameters are listed in the captions of these two tables. It is evident that the HLConcELM errors decrease approximately exponentially (before saturation) as the number of collocation points or the number of nodes in the neural network increases.

Case #2: No Exact Solution and Comparison with Chebfun. We next consider a case with no exact solution available, and so we use the numerical result obtained by Chebfun [14] as a reference to compare with the HLConcELM solution. We employ the following parameter values for the domain and the system (41),

$$a = -1, \quad b = 1, \quad t_f = 1, \quad f(x,t) = 0, \quad g(x) = \frac{7}{4}\left[\cos(\pi x) + 1\right].$$

Figure 33 illustrates the distributions of the locHLConcELM solution and the Chebfun solution for the real part, the imaginary part, and the norm of $h(x,t)$. The Chebfun solution is obtained on 1024 Fourier grid points along the x direction with a time step size $\Delta t = 10^{-4}$. For locHLConcELM with block time marching, we have employed 5 uniform time blocks, 3 sub-domains along the x direction within each time block (interior sub-domain boundaries located at $x = -0.35$ and 0.35), and a local neural network $\mathbf{M} = [2, 400, 2]$ with the Gaussian activation function on each sub-domain. The other simulation parameter values are listed in the figure caption. No apparent difference can be discerned between the locHLConcELM solution and the Chebfun solution qualitatively.

Figure 34 provides a comparison between the locHLConcELM solution and the Chebfun solution quantitatively. Figures 34(a,b,c) compare profiles of the

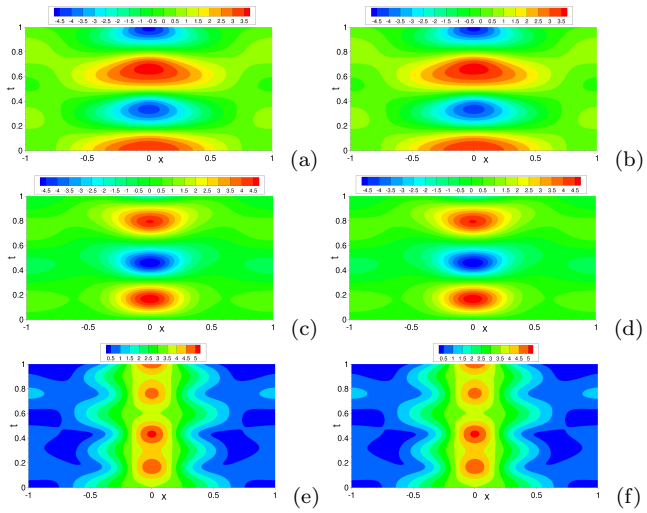

Fig. 33 (Appendix F) Schrodinger equation (case #2): Distributions of (a,b) the real part, (c,d) the imaginary part, and (e,f) the norm, of $h(x,t)$ obtained by HLConcELM (a,c,e) and by Chebfun (b,d,f). In Chebfun, 1024 Fourier grid points in x, time step size $\Delta t = 10^{-4}$. In HLConcELM, 5 uniform time blocks, 3 sub-domains along the x direction within each time block (sub-domain boundary points $\mathcal{X} = [-1, -0.35, 0.35, 1]$), local neural network $[2, 400, 2]$ and $Q = 25 \times 25$ uniform collocation points on each sub-domain, $\mathbf{R} = 2.0$, Gaussian activation function.

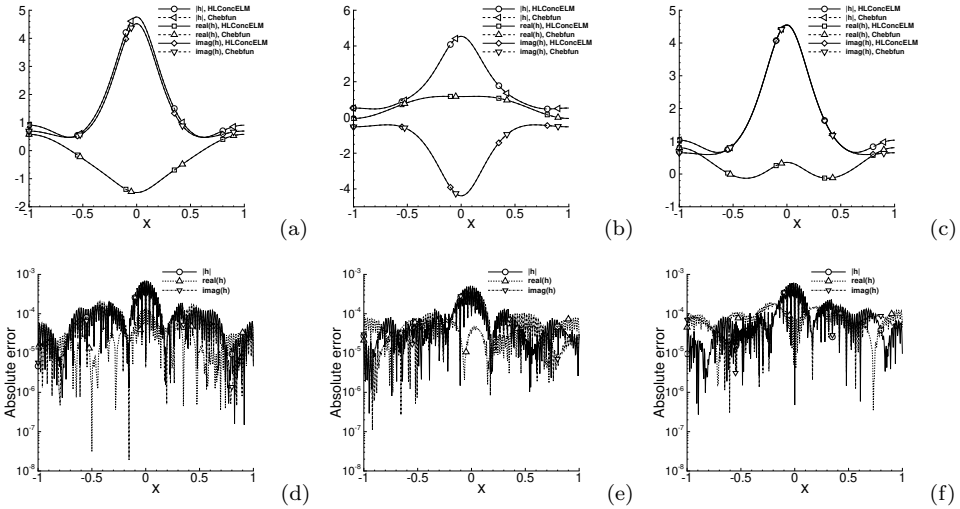

Fig. 34 (Appendix F) Schrodinger equation (case #2): Comparison of the solution profiles between locHLConcELM and Chebfun at (a) $t = 0.2$, (b) $t = 0.5$, and (c) $t = 0.8$. Profiles of the absolute error between the locHLConcELM solution and the Chebfun solution at (d) $t = 0.2$, (e) $t = 0.5$, and (f) $t = 0.8$. Simulation settings and parameters follow those of Figure 33.

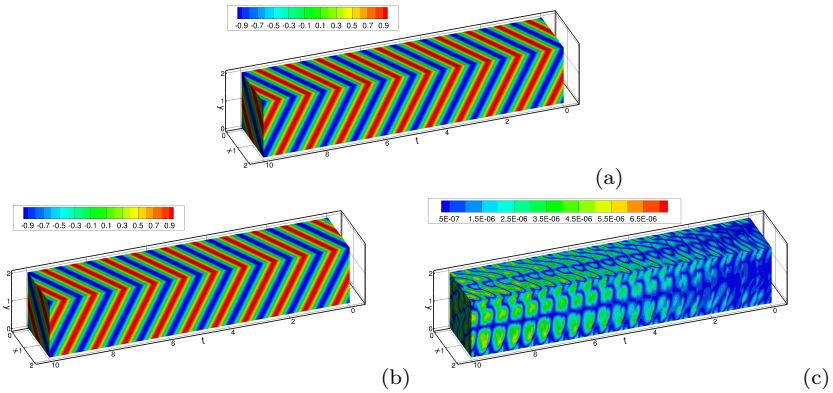

Fig. 35 (Appendix G) 2D Advection equation: Distributions of (a) the exact solution, (b) the HLConcELM solution and (c) its point-wise absolute error in the spatial-temporal domain. In HLConcELM (with block time marching), 20 uniform time blocks, neural network $[3, 1000, 1]$ and $Q = 15 \times 15 \times 15$ uniform collocation points within each time block, hidden magnitude vector $\mathbf{R} = 0.6$, Gaussian activation function.

collocation points	max error	rms error
$4 \times 4 \times 4$	1.87E+0	7.32E-1
$8 \times 8 \times 8$	1.59E-1	3.35E-2
$12 \times 12 \times 12$	9.92E-5	1.77E-5
$16 \times 16 \times 16$	7.94E-6	1.94E-6
$20 \times 20 \times 20$	8.29E-6	2.11E-6

Table 20 (Appendix G) 2D Advection equation: The maximum and rms errors of the HLConcELM solution versus the number of uniform collocation points. 20 uniform time blocks, neural network $[3, 1000, 1]$ within each time block, hidden magnitude vector $\mathbf{R} = 0.6$, Gaussian activation function.

locHLConcELM solution and the Chebfun solution for the real part, the imaginary part and the norm of $h(x, t)$ at three time instants $t = 0.2$, 0.5 and 0.8. The locHLConcELM solution profiles and the Chebfun solution profiles exactly overlap with one another. Figures 34(d,e,f) show profiles of the absolute error between the locHLConcELM solution and the Chebfun solution at the same time instants. One can observe that the difference between locHLConcELM and Chebfun is on the order of 10^{-4}, suggesting that the locHLConcELM result agrees well with the Chebfun result for this problem.

Appendix G. Two-Dimensional Advection Equation

This appendix provides a further test of the HLConcELM method with the advection equation in two spatial dimensions plus time. Note that the numerical results in Section 3.2 are for the one-dimensional advection equation (plus time). We consider the spatial-temporal domain $(x, y, t) \in \Omega = [0, 2] \times [0, 2] \times [0, 10]$, and

M	max error	rms error
200	8.51E-1	3.50E-1
400	3.75E-2	1.05E-2
600	1.62E-3	4.46E-4
800	1.79E-4	4.23E-5
1000	7.39E-6	1.90E-6

Table 21 (Appendix G) 2D Advection equation: The maximum and rms errors of the HLConcELM solution versus the number of nodes in hidden layer. 20 uniform time blocks, neural network $[3, M, 1]$ (M varied) and $Q = 15 \times 15 \times 15$ uniform collocation points within each time block, hidden magnitude vector $\mathbf{R} = 0.6$, Gaussian activation function.

the advection equation on Ω with periodic boundary conditions,

$$\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} - \frac{\partial u}{\partial y} = 0, \tag{43a}$$

$$u(0, y, t) = u(2, y, t), \quad u(x, 0, t) = u(x, 2, t), \tag{43b}$$

$$u(x, y, 0) = \cos\left[\pi\left(x + y - 1\right)\right]. \tag{43c}$$

This initial/boundary value problem has the following exact solution,

$$u(x, y, t) = \cos\left[\pi\left(x + y + t - 1\right)\right]. \tag{44}$$

We solve the problem (43) by the HLConcELM method together with block time marching (see Remark 5). We employ 20 uniform time blocks in the simulation, with a size 0.5 for each time block. Within each time block we employ a neural network architecture $[3, M, 1]$ (M varied) with the Gaussian activation function, and a uniform set of $Q = Q_1 \times Q_1 \times Q_1$ collocation points (Q_1 varied). The seed for the random number generators is set to 100 in the numerical tests. After the network is trained, we evaluate the neural network on another fixed set of $Q_{eval} = 51 \times 51 \times 51$ uniform grid points within each time block to attain the HLConcELM solution values for all time blocks. We then compare the HLConcELM solution with the exact solution (44) on the same set of Q_{eval} points within each time block, to compute the maximum (l^∞) and rms (l^2) errors over the entire spatial-temporal domain Ω. The error values as computed above are said to be associated with the neural network $[3, M, 1]$ with the $Q = Q_1 \times Q_1 \times Q_1$ collocation points.

Figure 35 illustrates the distributions of the exact solution (44), the HLConcELM solution, and the point-wise absolute error of the HLConcELM solution over Ω. The values for the simulation parameters in HLConcELM are provided in the figure caption. The HLConcELM solution is quite accurate, with the maximum error on the order 10^{-6} over the entire domain Ω.

The exponential convergence of the HLConcELM accuracy is illustrated by Tables 20 and 21. Table 20 lists the maximum and rms errors of HLConcELM (over Ω) as a function of the number of collocation points (Q), obtained with a neural network $[3, 1000, 1]$. Table 21 lists the maximum and rms errors of HLConcELM as a function of the number of hidden nodes (M) in the neural network, obtained on a fixed set of $Q = 15 \times 15 \times 15$ uniform collocation points. The other simulation parameter values are provided in the captions of these tables. It is evident that the HLConcELM errors decrease exponentially (before saturation) with increasing number of collocation points or increasing number of hidden nodes in the network.

References

1. Alaba, P., Popoola, S., Olatomiwa, L., Akanle, M., Ohunakin, O., Adetiba, E., Alex, O., Atayero, A., Daud, W.: Towards a more efficient and cost-sensitive extreme learning machine: a state-of-the-art review of recent trend. Neurocomputing **350**, 70–90 (2019)
2. Basdevant, C., Deville, M., Haldenwang, P., Lacroix, J., Ouazzani, J., Peyret, R., Orlandi, P., Patera, A.: Spectral and finite difference solutions of the Burgers equation. Computers and Fluids **14**, 23–41 (1986)
3. Braake, H., Straten, G.: Random activation weight neural net (RAWN) for fast non-iterative training. Eng. Applic. Artif. Intell. **8**, 71–80 (1995)
4. Branch, M., Coleman, T., Li, Y.: A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. SIAM Journal on Scientific Computing **21**, 1–23 (1999)
5. Byrd, R., Schnabel, R., Shultz, G.: Approximate solution of the trust region problem by minimization over two-dimensional subspaces. Math. Programming (1988)
6. Calabro, F., Fabiani, G., Siettos, C.: Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients. Computer Methods in Applied Mechanics and Engineering **387**, 114188 (2021)
7. Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., Yang, S.: Adanet: Adaptive structural learning of artificial neural networks. arXiv:1607.01097 (2016)
8. Cyr, E., Gulian, M., Patel, R., Perego, M., Trask, N.: Robust training and initialization of deep neural networks: An adaptive basis viewpoint. Proceedings of Machine Learning Research **107**, 512–536 (2020)
9. Dong, S., Li, Z.: 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)
10. Dong, S., Li, Z.: 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)
11. Dong, S., Ni, N.: A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. Journal of Computational Physics **435**, 110242 (2021). (also arXiv:2007.07442)
12. Dong, S., Yang, J.: Numerical approximation of partial differential equations by a variable projection method with artificial neural networks. arXiv:2201.09989 (2022)
13. Dong, S., Yang, J.: On computing the hyperparameter of extreme learning machines: algorithm and application to computational PDEs and comparison with classical and high-order finite elements. Journal of Computational Physics **463**, 111290 (2022). (also arXiv:2110.14121)
14. Driscoll, T., Hale, N., Trefethen, L.: Chebfun Guide. Pafnuty Publications, Oxford (2014)
15. Dwivedi, V., Srinivasan, B.: Physics informed extreme learning machine (pielm) − a rapid method for the numerical solution of partial differential equations. Neurocomputing **391**, 96–118 (2020)
16. Dwivedi, V., Srinivasan, B.: A normal equation-based extreme learning machine for solving linear partial differential equations. Journal of Computing and Information Science in Engineering **22**, 014502 (2022)
17. E, W., Yu, B.: The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics **6**, 1–12 (2018)
18. Fabiani, G., Calabro, F., Russo, L., Siettos, C.: Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines. Journal of Scientific Computing **89**, 44 (2021)
19. Fokina, D., Oseledets, I.: Growing axons: greedy learning of neural networks with application to function approximation. arXiv:1910.12686 (2020)
20. Freire, A., Rocha-Neto, A., Barreto, G.: On robust randomized neural networks for regression: a comprehensive review and evaluation. Neural Computing and Applications **32**, 16931–16950 (2020)
21. Galaris, E., Fabiani, G., Calabro, F., Serafino, D., Siettos, C.: Numerical solution of stiff ODEs with physics-informed random projection neural networks. arXiv:2108.01584 (2021)
22. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press (2016)
23. Guo, P., Chen, C., Sun, Y.: An exact supervised learning for a three-layer supervised neural network. In: Proceedings of 1995 International Conference on Neural Information Processing, pp. 1041–1044 (1995)

24. He, J., Xu, J.: MgNet: A unified framework for multigrid and convolutional neural network. Science China Mathematics **62**, 1331–1354 (2019)
25. Hendrycks, D., Gimpel, K.: Gaussian error linear units (GELU). arXiv:1606.08415 (2016)
26. Huang, G., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Transactions on Neural Networks **17**, 879–892 (2006)
27. Huang, G., Huang, G., Song, S., You, K.: Trends in extreme learning machines: a review. Neural Networks **61**, 32–48 (2015)
28. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.: Densely connected convolutional networks. arXiv:1608.06993 (2018)
29. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: 2004 IEEE International Joint Conference on Neural Networks, vol. 2, pp. 985–990 (2004)
30. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**, 489–501 (2006)
31. Igelnik, B., Pao, Y.: Stochastic choice of basis functions in adaptive function approximation and the functional-link net. IEEE Transactions on Neural Networks **6**, 1320–1329 (1995)
32. Jaeger, H., Lukosevicius, M., Popovici, D., Siewert, U.: Optimization and applications of echo state networks with leaky integrator neurons. Neural Networks **20**, 335–352 (2007)
33. Jagtap, A., Kharazmi, E., Karniadakis, G.: 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)
34. Karniadakis, G., Kevrekidis, G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed machine learning. Nature Reviews Physics **3**, 422–440 (2021)
35. Karniadakis, G., Sherwin, S.: Spectral/hp element methods for computational fluid dynamics, 2nd edn. Oxford University Press (2005)
36. Katuwal, R., Suganthan, P., Tanveer, M.: Random vector functional link neural network based ensemble deep learning. arXiv:1907.00350 (2019)
37. Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., Mahoney, M.: Characterizing possible failure modes in physics-informed neural networks. arXiv:2109.01050 (2021)
38. Kuramoto, Y.: Diffusion-induced chaos in reaction systems. Progress of Theoretical Physics Supplement **64**, 346–367 (1978)
39. Li, J.Y., Chow, W., Igelnik, B., Pao, Y.H.: Comments on "stochastic choice of basis functions in adaptive function approximaton and the functional-link net". IEEE Trans. Neural Netw. **8**, 452–454 (1997)
40. Liu, H., Xing, B., Wang, Z., Li, L.: 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)
41. Liu, M., Hou, M., Wang, J., Cheng, Y.: Solving two-dimensional linear partial differential equations based on Chebyshev neural network with extreme learning machine algorithm. Engineering Computations **38**, 874–894 (2021)
42. Lu, L., Meng, X., Mao, Z., Karniadakis, G.: DeepXDE: a deep learning library for solving differential equations. SIAM Review **63**, 208–228 (2021)
43. Lukosevicius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**, 127–149 (2009)
44. Maas, W., Markram, H.: On the computational power of recurrent circuits of spiking neurons. J. Comput. Syst. Sci. **69**, 593–616 (2004)
45. Needell, D., Nelson, A., Saab, R., Salanevich, P.: Random vector functional link networks for function approximation on manifolds. arXiv:2007.15776 (2020)
46. Nocedal, J., Wright, S.: Numerical Optimization, Second Edition. Springer (2006)
47. Panghal, S., Kumar, M.: Optimization free neural network approach for solving ordinary and partial differential equations. Engineering with Computers **37**, 2989–3002 (2021)
48. Pao, Y., Park, G., Sobajic, D.: Learning and generalization characteristics of the random vector functional-link net. Neurocomputing **6**, 163–180 (1994)
49. Pao, Y., Takefuji, Y.: Functional-link net computing: theory, system architecture, and functionalities. Computer **25**, 76–79 (1992)
50. Rahimi, A., Recht, B.: 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)
51. Raissi, M., Perdikaris, P., Karniadakis, G.: 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)

52. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**, 386–408 (1958)
53. Scardapane, S., Wang, D.: Randomness in neural networks: an overview. WIREs Data Mining Knowl. Discov. **7**, e1200 (2017)
54. Sirignano, J., Spoliopoulos, K.: DGM: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics **375**, 1339–1364 (2018)
55. Sivashinsky, G.: Nonlinear analysis of hydrodynamic instability in laminar flames – I. derivation of basic equations. Acta Astronautica **4**, 1177–1206 (1977)
56. Suhanthan, P., Katuwal, R.: On the origins of randomization-based feedforward neural networks. Applied Soft Computing **105**, 107239 (2021)
57. Sun, H., Hou, M., Yang, Y., Zhang, T., Weng, F., Han, F.: Solving partial differential equations based on bernsteirn neural network and extreme learning machine algorithm. Neural Processing Letters **50**, 1153–1172 (2019)
58. Tang, K., Wan, X., Liao, Q.: Adaptive deep density estimation for fokker-planck equations. Journal of Computational Physics **457**, 111080 (2022)
59. Verma, B., Mulawka, J.: A modified backpropagation algorithm. In: Proceedings of 1994 IEEE International Conference on Neural Networks, vol. 2, pp. 840–844 (1994)
60. Wan, X., Wei, S.: VAE-KRnet and its applications to variational Bayes. Communications in Computational Physics **31**, 1049–1082 (2022)
61. Wang, S., Yu, X., Perdikaris, P.: When and why PINNs fail to train: a neural tangent kernel perspective. Journal of Computational Physics **449**, 110768 (2022)
62. Wang, Y., Lin, G.: Efficient deep learning techniques for multiphase flow simulation in heterogeneous porous media. Journal of Computational Physics **401**, 108968 (2020)
63. Webster, C.: Alan Turing's unorganized machines and artificial neural networks: his remarkable early work and future possibilities. Evol. Intel. **5**, 35–43 (2012)
64. Widrow, B., Greenblatt, A., Kim, Y., Park, D.: The no-prop algorithm: a new learning algorithm for multilayer neural networks. Neural Networks **37**, 182–188 (2013)
65. Wilamowski, B., Yu, H.: Neural network learning without backpropagation. IEEE Transactions on Neural Networks **21**, 1793–1803 (2010)
66. Winovich, N., Ramani, K., Lin, G.: ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. Journal of Computational Physics **394**, 263–279 (2019)
67. Yang, Y., Hou, M., Luo, J.: 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)
68. Yang, Z., Dong, S.: 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 pp. 229–257 (2019)
69. Yang, Z., Dong, S.: A roadmap for discretely energy-stable schemes for dissipative systems based on a generalized auxiliary variable with guaranteed positivity. Journal of Computational Physics **404**, 109121 (2020). (also arXiv:1904.00141)
70. Yang, Z., Lin, L., Dong, S.: A family of second-order energy-stable schemes for Cahn-Hilliard type equations. Journal of Computational Physics **383**, 24–54 (2019)
71. Zhang, L., Suganthan, P.: A comprehensive evaluation of random vector functional link networks. Inf. Sci. **367–368**, 1094–1105 (2016)
72. Zheng, X., Dong, S.: An eigen-based high-order expansion basis for structured spectral elements. Journal of Computational Physics **230**, 8573–8602 (2011)

Statements and Declarations

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

Competing Interests:
The authors have no relevant financial or non-financial interests to disclose.

Author Contributions:
N. Ni: software, data acquisition, data visualization, data analysis, writing of paper.
S. Dong: conceptualization, methodology, software, data acquisition, data analysis, writing of paper.

Data Availability:
The datasets related to this paper are available from the correpsonding author on reasonable request.