Journal of Computational Physics 455 (2022) 111021

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Self-adaptive deep neural network: Numerical approximation to functions and PDEs ☆

Zhiqiang Cai [a,*], Jingshuang Chen [a], Min Liu [b]

[a] Department of Mathematics, Purdue University, 150 N. University Street, West Lafayette, IN 47907-2067, United States of America
[b] School of Mechanical Engineering, Purdue University, 585 Purdue Mall, West Lafayette, IN 47907-2088, United States of America

ARTICLE INFO

Article history:
Received 9 August 2021
Received in revised form 24 January 2022
Accepted 25 January 2022
Available online 29 January 2022

Keywords:
Self-adaptivity
Advection-reaction equation
Least-squares approximation
Deep neural network
ReLU activation

ABSTRACT

Designing an optimal deep neural network for a given task is important and challenging in many machine learning applications. To address this issue, we introduce a self-adaptive algorithm: the adaptive network enhancement (ANE) method, written as loops of the form

train → **estimate** → **enhance**.

Starting with a small two-layer neural network (NN), the step **train** is to solve the optimization problem at the current NN; the step **estimate** is to compute *a posteriori* estimator/indicators using the solution at the current NN; the step **enhance** is to add new neurons to the current NN.

Novel network enhancement strategies based on the computed estimator/indicators are developed in this paper to determine how many new neurons and when a new layer should be added to the current NN. The ANE method provides a natural process for obtaining a good initialization in training the current NN; in addition, we introduce an advanced procedure on how to initialize newly added neurons for a better approximation. We demonstrate that the ANE method can automatically design a nearly minimal NN for learning functions exhibiting sharp transitional layers as well as discontinuous solutions of hyperbolic partial differential equations.

© 2022 Elsevier Inc. All rights reserved.

1. Introduction

Deep neural network (DNN) has achieved astonishing performance in computer vision, natural language processing, and many other artificial intelligence (AI) tasks (see, e.g., [8,16,12]). This success encourages wide applications to other fields, including recent studies of using DNN models to learn solutions of partial differential equations (PDEs) (see, e.g., [2,7,6,10,20,23]). The phenomenal performance on many AI tasks comes at the cost of high computational complexity. Accordingly, designing efficient network architectures for DNN is an important step towards enabling the wide deployment of DNN in various applications.

Studies and applications of neural network (NN) may be traced back to the work of Hebb [14] in the late 1940's and Rosenblatt [21] in the 1950's. DNN produces a new class of functions through compositions of linear transformations and

☆ This work was supported in part by the National Science Foundation under grant DMS-2110571.
* Corresponding author.
E-mail addresses: caiz@purdue.edu (Z. Cai), chen2042@purdue.edu (J. Chen), liu66@purdue.edu (M. Liu).

https://doi.org/10.1016/j.jcp.2022.111021
0021-9991/© 2022 Elsevier Inc. All rights reserved.

activation functions. This class of functions is extremely rich. For example, it contains piece-wise polynomials, which are the footing of spectral elements, and continuous and discontinuous finite element methods for computer simulations of complex physical, biological, and human-engineered systems. It approximates polynomials of any degree with exponential efficiency, even using simple activation functions like ReLU. More importantly, a neural network function can automatically adapt to a target function or the solution of a PDE.

Despite great successes of DNN in many practical applications, it is widely accepted that approximation properties of DNN are not yet well understood and that understanding on why and how they work could lead to significant improvements in many machine learning applications. First, some empirical observations suggest that deep network can approximate many functions more accurately than shallow network, but rigorous study on the theoretical advantage of deep network is scarce. Therefore, even in the manual design of network models, the addition of neurons along depth or width is *ad-hoc*. Second, current methods on design of the architecture of DNN in terms of their width and depth are empirical. Tuning of depth and width is tedious, mainly from experimental results in ablation studies which typically require domain knowledge about the underlying problem. Third, there is a tendency in practice to use over-parametrized neural networks; this leads to a high-dimensional nonlinear optimization problem which is much more difficult to train than a low-dimensional one. These considerations suggest that a fundamental, open question to be addressed in scientific machine learning is: what is the optimal network model required, in terms of width, depth, and the number of parameters, to learn data, a function, or the solution of a PDE within some prescribed accuracy?

To address this issue, we introduce a self-adaptive algorithm: the adaptive network enhancement (ANE) method, written as loops of the form

$$\textbf{train} \rightarrow \textbf{estimate} \rightarrow \textbf{enhance}.$$

Starting with a small two-layer NN, the step **train** is to solve the optimization problem of the current NN; the step **estimate** is to compute *a posteriori* estimator/indicators using the solution at the current NN; the step **enhance** is to add new neurons to the current NN. This adaptive algorithm learns not only from given information (data, function, PDE) but also from the current computer simulation, and it is therefore a learning algorithm at a level which is more advanced than common machine learning algorithms.

To develop an efficient ANE method, we need to address the following essential questions at each adaptive step when the current NN is not sufficient for the given task:

(a) how many new neurons should be added?
(b) when should a new layer be added?

For a two-layer NN, we proposed the ANE method (see Algorithm 4.1) for learning a given function in [18] and the solution of a given self-adjoint elliptic PDEs through the Ritz formulation in [17]. In the case of a two-layer NN, question (b) is irrelevant and question (a) was addressed by introducing a network enhancement strategy that decides the number of new neurons to be added in the first hidden layer. This strategy is based on the physical partition of the current computer simulation determined by the *a posteriori* error indicators (see Algorithm 3.1).

For a multi-layer NN, it is challenging to address both the questions. First, the role of a neuron in approximation at a hidden layer varies and depends on which hidden layer the neuron is located. Second, there is almost no understanding on the role of a specific hidden layer in approximation and, hence, we have no *a priori* approximation information for determining when a new layer should be added. To resolve question (a) for a multi-layer, we will exploit the geometric property of the current computer simulation and introduce a novel enhancement strategy (see Algorithm 4.2) that determines the number of new neurons to be added at a hidden layer other than the first hidden layer. For question (b), we will introduce a computable quantity to measure the improvement rate of two consecutive NNs per the relative increase of parameters. When the improvement rate is small, then a new layer is started.

Training DNN, i.e., determining the values of the parameters of DNN, is a problem in nonlinear optimization. This high dimensional, nonlinear optimization problem tends to be computationally intensive and complicated and is usually solved iteratively by the method of gradient descent and its variations (see [4]). In general, a nonlinear optimization has many solutions, and the desired one is obtained only if we start from a close enough first approximation. A common way to obtain a good initialization is by the method of continuation [1]. The ANE method provides a natural process for obtaining a good initialization. Basically, the approximation at the previous NN is already a good approximation to the current NN in the loops of the ANE method. To provide a better approximation, we initialize the weights and bias of newly added neurons at the first hidden layer by using the physical partition of the domain (see Section 3 and [18] for details); in this paper we introduce an advanced procedure on how to initialize newly added neurons at a hidden layer that is not the first hidden layer.

For simplicity of presentation, the ANE method for a multi-layer NN is first described for learning a given function through the least-squares loss function (see Section 4). The method is then applied to learn solutions of linear advection-reaction equations through the least-squares neural network (LSNN) method introduced in [5] (see Section 7). We demonstrate that the ANE method can automatically design a nearly minimal NN for learning functions exhibiting sharp transitional layers as well as discontinuous solutions of hyperbolic PDEs.

Recently, there is growing interest in automatic machine learning (AutoML) in an effort of replacing a manual design process of architectures by human experts. Neural architecture search (NAS) method (see a survey paper [11] and reference therein) presents a general methodology at high level on AutoML. It consists of three components: search space, search strategy, and performance estimation strategy. Usually the resulting algorithm is computationally intensive because it explores a wide range of potential network architectures. Nevertheless, the NAS outperforms manually designed architectures in accuracy on some tasks such as image classification, object detection, or semantic segmentation. The NAS based on the physics-informed neural network is recently used for solving stochastic groundwater flow problem in [13].

The paper is organized as follows. DNN, the best least-squares (LS) approximation to a given function using DNN, and the discrete counterpart of the best LS approximation are introduced in Section 2. The physical partition for a DNN function is described in Section 3. The ANE method, initialization of parameters at different stage, and numerical experiments are presented in Sections 4, 5 and 6, respectively. Finally, application of the ANE method to the linear advection-reaction equation is given in Section 7.

2. Deep neural network and least-squares approximation

A deep neural network defines a function of the form

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \boldsymbol{\omega}^L \cdot \left(N^{(L-1)} \circ \cdots \circ N^{(1)}(\mathbf{x})\right) - b^L : \mathbf{x} \in \mathbb{R}^d \longrightarrow \mathbf{y} = \mathcal{N}(\mathbf{x}) \in \mathbb{R}, \tag{2.1}$$

where d is the dimension of input $\mathbf{x}$, $\boldsymbol{\omega}^{(L)} \in \mathbb{R}^{n_{L-1}}$, $b^{(L)} \in \mathbb{R}$, the symbol $\circ$ denotes the composition of functions, and L is the depth of the network. For $l = 1, \cdots, L-1$, the $N^{(l)} : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$ is called the l^{th} hidden layer of the network defined by

$$N^{(l)}(\mathbf{x}^{(l-1)}) = \sigma(\boldsymbol{\omega}^{(l)}\mathbf{x}^{(l-1)} - \mathbf{b}^{(l)}) \quad \text{for } \mathbf{x}^{(l-1)} \in \mathbb{R}^{n_{l-1}}, \tag{2.2}$$

where $\boldsymbol{\omega}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$, $\mathbf{x}^{(0)} = \mathbf{x}$, and $\sigma(t) = \max\{0, t\}^p$ with positive integer p is the activation function and its application to a vector is defined component-wise. This activation function is referred to as a spline activation ReLU^p. When $p = 1$, $\sigma(t)$ is the popular rectified linear unit (ReLU). There are many other activation functions such as (logistic, Gaussian, arctan) sigmoids (see, e.g., [19]).

Let $\boldsymbol{\theta}$ denote all parameters to be trained, i.e., the weights $\{\boldsymbol{\omega}^{(l)}\}_{l=1}^L$ and the bias $\{\mathbf{b}^{(l)}\}_{l=1}^L$. Then the total number of parameters is given by

$$N = M_d(L) = \sum_{l=1}^{L} n_l \times (n_{l-1} + 1). \tag{2.3}$$

Denote the set of all DNN functions by

$$\mathcal{M}_N(\boldsymbol{\theta}, L) = \left\{\boldsymbol{\omega}^L \cdot \left(N^{(L-1)} \circ \cdots \circ N^{(1)}(\mathbf{x})\right) - b^L : \boldsymbol{\omega}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}},\ \mathbf{b}^{(l)} \in \mathbb{R}^{n_l} \text{ for } l = 1, ..., L-1\right\}.$$

Let $f(\mathbf{x}) \in \mathbb{R}$ be a given target function defined in a domain $\Omega \in \mathbb{R}^d$. Training DNN to learn the function $f(\mathbf{x})$ using least-squares loss function amounts to solve the following best least-squares approximation: find $f_N(\mathbf{x}; \boldsymbol{\theta}^*) \in \mathcal{M}_N(\boldsymbol{\theta}, L)$ such that

$$\|f(\cdot) - f_N(\cdot; \boldsymbol{\theta}^*)\| = \min_{v \in \mathcal{M}_N(\boldsymbol{\theta}, L)} \|f - v\| = \min_{\boldsymbol{\theta} \in \mathbb{R}^N} \|f(\cdot) - v(\cdot; \boldsymbol{\theta})\|, \tag{2.4}$$

where $v(\mathbf{x}; \boldsymbol{\theta}) \in \mathcal{M}_N(\boldsymbol{\theta}, L)$ is of the form

$$v(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\omega}^L \cdot \left(N^{(L-1)} \circ \cdots \circ N^{(1)}(\mathbf{x})\right) - b^L$$

with $N^{(l)}(\cdot)$ defined in (2.2), and $\|v(\cdot)\| = \left(\int_\Omega v^2(\mathbf{x})\, d\mathbf{x}\right)^{1/2}$ is the $L^2(\Omega)$ norm.

Let $\mathcal{I}$ be the integral operator over the domain Ω given by

$$\mathcal{I}(f) = \int_\Omega f(\mathbf{x})\, d\mathbf{x}. \tag{2.5}$$

Let $\mathcal{T} = \{K : K \text{ is an open subdomain of } \Omega\}$ be a partition of the domain Ω, i.e., union of all subdomains of $\mathcal{T}$ equals to the whole domain Ω and that any two distinct subdomains of $\mathcal{T}$ have no intersection. Let $\mathcal{Q}_\mathcal{T}$ be a quadrature operator based on the partition $\mathcal{T}$, i.e., $\mathcal{I}(v) \approx \mathcal{Q}_\mathcal{T}(v)$, such that

$$\|v\|_\mathcal{T} = \sqrt{(v, v)_\mathcal{T}} = \sqrt{\mathcal{Q}_\mathcal{T}(v^2)}$$

defines a weighted l_2-norm. The best discrete least-squares approximation with numerical integration over the partition $\mathcal{T}$ is to find $f_\mathcal{T}(\mathbf{x}; \boldsymbol{\theta}^*_\mathcal{T}) \in \mathcal{M}_N(\boldsymbol{\theta}, L)$ such that

$$\|f(\cdot) - f_{\mathcal{T}}(\cdot;\boldsymbol{\theta}^*_{\mathcal{T}})\|_{\mathcal{T}} = \min_{v\in\mathcal{M}_N(\boldsymbol{\theta},L)} \|f - v\|_{\mathcal{T}} = \min_{\boldsymbol{\theta}\in\mathbb{R}^N} \|f(\cdot) - v(\cdot;\boldsymbol{\theta})\|_{\mathcal{T}}. \tag{2.6}$$

Theorem 2.1. *Assume that there exists a positive constant α such that $\alpha\|v\|^2 \le \|v\|^2_{\mathcal{T}}$ for all $v \in \mathcal{M}_{2N} \equiv \mathcal{M}_N(\boldsymbol{\theta}, L) \oplus \mathcal{M}_N(\boldsymbol{\theta}, L)$. Let $f_{\mathcal{T}}$ be a solution of (2.6). Then there exists a positive constant C such that*

$$C\|f - f_{\mathcal{T}}\| \le \inf_{v\in\mathcal{M}_N(\boldsymbol{\theta},L)} \left\{ \|f - v\| + \sup_{w\in\mathcal{M}_{2N}} \frac{|(\mathcal{I} - \mathcal{Q}_{\mathcal{T}})(vw)|}{\|w\|} \right\} + \sup_{w\in\mathcal{M}_{2N}} \frac{|(\mathcal{I} - \mathcal{Q}_{\mathcal{T}})(fw)|}{\|w\|}. \tag{2.7}$$

Proof. The theorem may be proved in a similar fashion as that of Theorem 4.1 in [18]. □

3. Physical partition (PP)

As seen in [18], the physical partition of the current NN approximation plays a critical role in the ANE method for a two-layer NN. As we shall see, it is essential for our self-adaptive multi-layer NN as well. For simplicity of presentation, we consider ReLU activation function only in this section. The idea of our procedure for determining the physical partition can be easily extended to other activation functions even though the corresponding geometry becomes complex.

For any function $v \in \mathcal{M}_N(\boldsymbol{\theta}, k)$ with $k \ge 2$, it is easy to see that v is a continuous piece-wise linear function with respect to a partition $\mathcal{K}^{(k-1)}$ of the domain Ω. This partition is referred to as the *physical partition* of the function v in $\mathcal{M}_N(\boldsymbol{\theta}, k)$. This section describes how to determine the physical partition $\mathcal{K}^{(k-1)}$ of a function in $\mathcal{M}_N(\boldsymbol{\theta}, k)$. To this end, for $l = 1, \cdots, k-1$, denote by $\mathcal{K}^{(l)}$ the physical partition of the first l layers.

To determine the physical partition $\mathcal{K}^{(1)}$, notice that a two-layer NN with n_1 neurons generates the following set of functions:

$$\mathcal{M}_N(\boldsymbol{\theta}, 2) = \left\{ \sum_{i=1}^{n_1} \omega_i^{(2)} \sigma(\boldsymbol{\omega}_i^{(1)} \cdot \mathbf{x} - b_i^{(1)}) - b^{(2)} \,:\, \omega_i^{(2)},\, b_i^{(1)},\, b^{(2)} \in \mathbb{R},\ \boldsymbol{\omega}_i^{(1)} \in \mathcal{S}^{d-1} \right\}, \tag{3.1}$$

where $\mathcal{S}^{d-1}$ is the unit sphere in $\mathbb{R}^d$. The $\mathcal{M}_N(\boldsymbol{\theta}, 2)$ may be viewed as an extension of the free-knot spline functions [22] to multi-dimension, and its free breaking hyper-planes are

$$\mathcal{P}_j :\ \boldsymbol{\omega}_j^{(1)} \cdot \mathbf{x} - b_j^{(1)} = 0 \quad \text{for } j = 1, \ldots, n_1. \tag{3.2}$$

This suggests that the physical partition $\mathcal{K}^{(1)}$ is formed by the boundary of the domain Ω and the hyper-planes $\left\{\boldsymbol{\omega}_j^{(1)} \cdot \mathbf{x} - b_j^{(1)} = 0\right\}_{j=1}^{n_1}$.

Next, we describe how to form the physical partition $\mathcal{K}^{(l)}$. Our procedure is based on the observation that for $l = 2, \ldots, k-1$, the $\mathcal{K}^{(l)}$ may be viewed as a refinement of the $\mathcal{K}^{(l-1)}$. For $j = 1, \cdots, n_l$, denote the function generated by the j^{th} neuron at the l^{th}-layer without the activation function by

$$g_j^{(l)}(\mathbf{x}) = \sum_{i=1}^{n_{l-1}} \omega_{ij}^{(l)} \sigma\left(\boldsymbol{\omega}_i^{(l-1)} \cdot \mathbf{x}^{(l-2)} - b_i^{(l-1)}\right) - b_j^{(l)},$$

where $\mathbf{x}^{(l-2)} = N^{(l-2)} \circ \cdots \circ N^{(1)}(\mathbf{x})$. It is clear that the functions $g_j^{(l)}(\mathbf{x})$ are continuous piece-wise linear functions with respect to the physical partition $\mathcal{K}^{(l-1)}$. The action of the activation function on $g_j^{(l)}(\mathbf{x})$, i.e., $\sigma(g_j^{(l)}(\mathbf{x})) = \max\{0, g_j^{(l)}(\mathbf{x})\}$ for $j = 1, \cdots, n_l$, generate n_l continuous piece-wise linear functions with respect to a refined partition of $\mathcal{K}^{(l-1)}$. Therefore, the refinement is created by the activation function through replacing negative values of $g_j^{(l)}(\mathbf{x})$ by zero (see Fig. 1 for illustration). In other words, the refinement is done by all new hyper-planes satisfying

$$g_j^{(l)}(\mathbf{x}) = 0 \quad \text{for } \ j = 1, \cdots, n_l. \tag{3.3}$$

To determine whether or not an element $K \in \mathcal{K}^{(l-1)}$ is refined, for each $g_j^{(l)}(\mathbf{x})$, we compute its values at vertices of K. If these values change signs, then the element K is partitioned by the hyper-plane $g_j^{(l)}(\mathbf{x}) = 0$ into two subdomains. It is possible that an element $K \in \mathcal{K}^{(l-1)}$ may be partitioned by many hyper-planes in (3.3). Denote the collection of refined elements in $\mathcal{K}^{(l-1)}$ by

$$\mathcal{K}_r^{(l-1)} = \left\{ K \in \mathcal{K}^{(l-1)} \,\middle|\, \exists\, j_0 \text{ such that values of } g_{j_0}^{(l)}(\mathbf{x}) \text{ at vertices of } K \text{ change signs} \right\}. \tag{3.4}$$

Denote by $\mathcal{K}_K^{(l)}$ the physical partition of element $K \in \mathcal{K}^{(l-1)}$ by hyper-planes in (3.3) and the boundary of K. Then the physical partition $\mathcal{K}^{(l)}$ by the first l hidden layers is given by

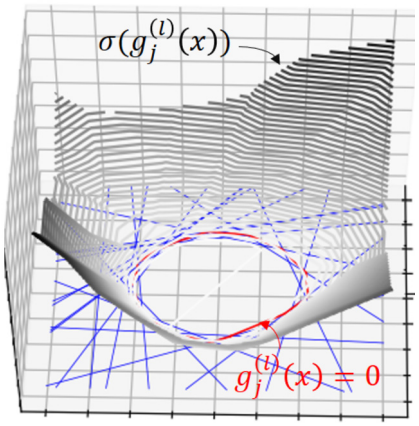


Fig. 1. Breaking lines generated by the j^{th} neuron of the l^{th}-layer.

$$\mathcal{K}^{(l)} = \left(\bigcup_{K \in \mathcal{K}_r^{(l-1)}} \mathcal{K}_K^{(l)} \right) \bigcup \left(\mathcal{K}^{(l-1)} \setminus \mathcal{K}_r^{(l-1)} \right). \tag{3.5}$$

The procedure of determining the physical partition of a function in $\mathcal{M}_N(\boldsymbol{\theta}, k)$ with $k \geq 3$ is summarized in Algorithm 3.1.

Algorithm 3.1 Physical partition.

For any function $v \in \mathcal{M}_N(\boldsymbol{\theta}, k)$ with $k \geq 3$, the partition $\mathcal{K}^{(1)}$ is determined by the boundary of the domain Ω and the hyper-planes $\left\{ \boldsymbol{\omega}_i^{(1)} \cdot \mathbf{x} - b_i^{(1)} = 0 \right\}_{i=1}^{n_1}$. For $l = 2, \cdots, k-1$,

(1) evaluate $g_j^{(l)}(\mathbf{x})$ at vertices of $\mathcal{K}^{(l-1)}$ for $j = 1, \cdots, n_l$;
(2) determine $\mathcal{K}_r^{(l-1)}$ using (3.4);
(3) for each $K \in \mathcal{K}_r^{(l-1)}$, determine its refinement by hyper-planes whose values change signs at vertices of K;
(4) $\mathcal{K}^{(l)}$ is given in (3.5).

4. Adaptive network enhancement method

Given a target function $f(\mathbf{x})$ and a prescribed tolerance $\epsilon > 0$ for approximation accuracy, in [18] we proposed the adaptive network enhancement (ANE) method for generating a two-layer ReLU NN and a numerical integration mesh such that

$$\| f(\cdot) - f_{\mathcal{T}}(\cdot; \boldsymbol{\theta}_{\mathcal{T}}^*) \| \leq \epsilon \, \| f \|, \tag{4.1}$$

where $f_{\mathcal{T}}(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{T}}^*)$ is the solution of the optimization problem in (2.6) over a two-layer NN with numerical integration defined on the partition $\mathcal{T}$.

For the convenience of readers and needed notations, we state Algorithm 5.1 of [18] (see Algorithm 4.1 below) for the case that numerical integration based on a partition $\mathcal{T}$ is sufficiently accurate. In this algorithm, $\mathcal{K}$ is the physical partition of the current approximation $f_{\mathcal{T}}$, $\xi_K = \| f - f_{\mathcal{T}} \|_{K,\mathcal{T}}$ is the local error in the physical subdomain $K \in \mathcal{K}$, and the network enhancement strategy introduced in [18] consists of either the average marking strategy:

$$\hat{\mathcal{K}} = \left\{ K \in \mathcal{K} : \xi_K \geq \frac{1}{\#\mathcal{K}} \sum_{K \in \mathcal{K}} \xi_K \right\}, \tag{4.2}$$

where $\#\mathcal{K}$ is the number of elements of $\mathcal{K}$, or the bulk marking strategy: finding a minimal subset $\hat{\mathcal{K}}$ of $\mathcal{K}$ such that

$$\sum_{K \in \hat{\mathcal{K}}} \xi_K^2 \geq \gamma_1 \sum_{K \in \mathcal{K}} \xi_K^2 \quad \text{for } \gamma_1 \in (0, 1). \tag{4.3}$$

With the subset $\hat{\mathcal{K}}$, the number of new neurons to be added to the current NN is equal to $\#\hat{\mathcal{K}}$, the number of elements in $\hat{\mathcal{K}}$.

For continuous functions exhibiting intersecting interface singularities or sharp transitional layer like discontinuities, numerical results in [18] showed the efficacy of the ANE method for generating a nearly minimal two-layer NN to learn

Algorithm 4.1 Adaptive Network Enhancement for a two-layer NN with a fixed $\mathcal{T}$.

Given a target function $f(\mathbf{x})$ and a tolerance $\epsilon > 0$, starting with a two-layer ReLU NN with a small number of neurons,

(1) solve the optimization problem in (2.6);
(2) estimate the total error by computing $\xi = \left(\sum_{K\in\mathcal{K}} \xi_K^2\right)^{1/2} / \|f\|_{\mathcal{T}}$;
(3) if $\xi < \epsilon$, then stop; otherwise, go to Step (4);
(4) add $\#\hat{\mathcal{K}}$ neurons to the network, then go to Step (1).

the target function within the prescribed accuracy. However, in the case that the transitional layer is over a circle but not a straight line, the approximation by a two-layer NN with a large number of neurons exhibits a certain level of oscillation; while the approximation using a three-layer NN with a small number of parameters is more accurate than the former and has no oscillation. Those numerical experiments suggest that a three-layer NN is needed for learning certain type of functions even if they are continuous.

In this section, we develop the ANE method for a multi-layer NN. To address question (b), i.e., when to add a new layer, we introduce a computable quantity denoted by η_r measuring the improvement rate of two consecutive NNs per the relative increase of parameters. If the improvement rate η_r is less than or equal to a prescribed expectation rate $\delta \in (0, 2)$, i.e.,

$$\eta_r \le \delta, \tag{4.4}$$

for two consecutive ANE runs, then the ANE method adds a new layer. Otherwise, the ANE adds neurons to the last hidden layer of the current network. Here a conservative strategy for adding a new layer is adopted by a double-run to check if inefficiency is identified when enhancing neurons in the current layer.

To define the improvement rate, denote the two consecutive NNs by $\mathcal{M}_{N^{\text{new}}}$ and $\mathcal{M}_{N^{\text{old}}}$, where the subscripts N^{new} and N^{old} are the number of parameters of these two NNs, respectively. Assume that the former is obtained by adding neurons in the last hidden layer of the latter. Let ξ^{new} and ξ^{old} be the error estimators of the approximations using $\mathcal{M}_{N^{\text{new}}}$ and $\mathcal{M}_{N^{\text{old}}}$, respectively. The improvement rate η_r is defined as

$$\eta_r = \left(\frac{\xi^{\text{old}} - \xi^{\text{new}}}{\xi^{\text{old}}}\right) \Big/ \left(\frac{(N^{\text{new}})^r - (N^{\text{old}})^r}{(N^{\text{new}})^r}\right),$$

where r is the order of the approximation with respect to the number of parameters and may depend on the activation function and the layer.

To determine the number of new neurons to be added in the last (but not first) hidden layer, our network enhancement strategy starts with the marked subset $\hat{\mathcal{K}}$ of $\mathcal{K}$ as in the first layer, where $\mathcal{K}$ is the physical partition of the current approximation. The subset $\hat{\mathcal{K}}$ is further regrouped into a new set $\mathcal{C} = \{C : C \text{ is a connected, open subdomain of } \Omega\}$ such that each element of $\mathcal{C}$ is either an isolated subdomain in $\hat{\mathcal{K}}$ or a union of connected subdomains in $\hat{\mathcal{K}}$. Now, the number of new neurons to be added equals to the number of elements in $\mathcal{C}$. This strategy is based on the observation that a multi-layer NN is capable of generating piece-wise breaking hyper-planes in connected subdomains by one neuron. Summarizing the above discussion, our network enhancement strategy on adding neurons and layers is described in Algorithm 4.2.

Algorithm 4.2 Network enhancement strategy.

Given an error estimator ξ and the improvement rate η_r,

(1) if (4.4) holds for two consecutive ANE runs, add a new hidden layer; otherwise, go to Step (2);
(2) use the marking strategy in (4.2) or (4.3) to generate $\hat{\mathcal{K}}$, and regroup $\hat{\mathcal{K}}$ to get $\mathcal{C}$;
(3) if there is only one hidden layer, add $\#\hat{\mathcal{K}}$ neurons to the first hidden layer; otherwise, add $\#\mathcal{C}$ neurons to the last hidden layer.

Assume that numerical integration on $\mathcal{T}$ is accurate, then the ANE method for generating a nearly minimal multi-layer neural network is described in Algorithm 4.3.

Algorithm 4.3 Adaptive network enhancement for a multi-layer NN with a fixed $\mathcal{T}$.

Given a target function $f(\mathbf{x})$ and a tolerance $\epsilon > 0$ for accuracy, starting with a two-layer NN with a small number of neurons and using one loop of Algorithm 4.1 to generate a two-layer NN, then

(1) solve the optimization problem in (2.6);
(2) estimate the total error by computing $\xi = \left(\sum_{K\in\mathcal{K}} \xi_K^2\right)^{1/2} / \|f\|_{\mathcal{T}}$;
(3) if $\xi < \epsilon$, then stop; otherwise, go to Step (4);
(4) compute the improvement rate η_r;
(5) add a new hidden layer or new neurons to the last hidden layer by Algorithm 4.2, then go to Step (1).

5. Initialization of training (iterative solvers)

To determine the values of the network parameters, we need to solve the optimization problem in (2.6), which is non-convex and, hence, computationally intensive and complicated. Currently, this problem is often solved by iterative optimization methods such as gradient descent (GD), Stochastic GD, Adam, etc. (see, e.g., [4] for a review paper in 2018 and references therein). Since non-convex optimizations usually have many solutions and/or many local minimum, it is then critical to start with a good initial guess in order to obtain the desired solution. As seen in [18], the ANE method itself is a natural continuation process for generating good initializations. In this section, we discuss initialization strategies of the ANE method for a multi-layer NN in two dimensions. Extensions to three dimensions are straightforward conceptually but more complicated algorithmically.

There are three cases that need to be initialized: (1) the beginning of the ANE method for a two-layer NN with a small number of neurons; (2) adding new neurons at the first layer; and (3) adding new neurons at the last hidden layer which is not the first layer. Initialization for both cases (1) and (2) was introduced in Section 5 of [18]. For the convenience of readers, we briefly describe them below.

The ANE method starts with a two-layer NN with n_1 neurons. Denote the input weights and bias by $\boldsymbol{\omega}^{(1)} = \left(\boldsymbol{\omega}_1^{(1)}, ..., \boldsymbol{\omega}_{n_1}^{(1)}\right)^T$ and $\mathbf{b}^{(1)} = \left(b_1^{(1)}, ..., b_{n_1}^{(1)}\right)^T$, respectively; and the output bias and weights by $\mathbf{c}^{(1)} = \left(b^{(2)}, \omega_1^{(2)}, ..., \omega_{n_1}^{(2)}\right)^T$. The initials of $\boldsymbol{\omega}^{(1)}$ and $\mathbf{b}^{(1)}$ are chosen such that the hyper-planes

$$\mathcal{P}_i : \boldsymbol{\omega}_i^{(1)} \cdot \mathbf{x} - b_i^{(1)} = 0 \quad \text{for } i = 1, ..., n_1$$

partition the domain uniformly. With the initial $\boldsymbol{\theta}^{(1)} = (\boldsymbol{\omega}^{(1)}, \mathbf{b}^{(1)})$ prescribed above, let

$$\varphi_0^{(1)}(\mathbf{x}) = 1 \quad \text{and} \quad \varphi_i^{(1)}(\mathbf{x}) = \sigma\left(\boldsymbol{\omega}_i^{(1)} \cdot \mathbf{x} - b_i^{(1)}\right) \quad \text{for } i = 1, ..., n_1.$$

Then the initial of $\mathbf{c}^{(1)}$ is given by the solution of the following system of linear algebraic equations

$$\boldsymbol{M}(\boldsymbol{\theta}^{(1)})\, \mathbf{c}^{(1)} = F(\boldsymbol{\theta}^{(1)}), \tag{5.1}$$

where the coefficient matrix $\boldsymbol{M}(\boldsymbol{\theta}^{(1)})$ and the right-hand side vector $F(\boldsymbol{\theta}^{(1)})$ are given by

$$\boldsymbol{M}\left(\boldsymbol{\theta}^{(1)}\right) = \left(\left(\varphi_j^{(1)}(\mathbf{x}), \varphi_i^{(1)}(\mathbf{x})\right)\right)_{(n_1+1)\times(n_1+1)} \quad \text{and} \quad F\left(\boldsymbol{\theta}^{(1)}\right) = \left(\left(f, \varphi_i^{(1)}(\mathbf{x})\right)\right)_{(n_1+1)\times 1},$$

respectively.

When adding new neurons at the first layer, the parameters associated with the old neurons will inherit the current approximation as their initials and those of the new neurons are initialized through the corresponding hyper-planes. Each new neuron is related to a sub-domain $K \in \hat{\mathcal{K}}$ (see Algorithm 4.1) and is initialized by setting its corresponding hyper-plane to pass through the centroid of K and orthogonal to the direction vector with the smallest variance of quadrature points in K. For details, see Section 5 of [18].

In the case (3), new neurons are added either at a new layer or at the current but not the first layer. As in the case (2), the parameters of the old neurons are initialized with their current approximations. Below we describe our strategy on how to initialize newly added neurons. First, consider the case in which we add neurons to start a new layer. Assume that the current NN has $k-1$ hidden layers. By Algorithm 4.2, the number of new neurons to be added at the k^{th} hidden layer equals to the number of elements in $\mathcal{C}^{(k-1)}$. For each element $C \in \mathcal{C}^{(k-1)}$, one neuron is added to the k^{th} hidden layer, and its output weight is randomly initialized. Below we introduce a strategy to initialize its bias and weights, $\boldsymbol{\omega}^{(k)} = \left(b^{(k)}, \omega_1^{(k)}, \cdots, \omega_{n_{k-1}}^{(k)}\right)^T = \left(\omega_0^{(k)}, \omega_1^{(k)}, \cdots, \omega_{n_{k-1}}^{(k)}\right)^T$. To this end, let us introduce a corresponding output function of the neuron to be added to refine C,

$$l_C(\mathbf{x}) = b^{(k)} + \sum_{i=1}^{n_{k-1}} \omega_i^{(k)} \sigma\left(\boldsymbol{\omega}_i^{(k-1)} \cdot \mathbf{x}^{(k-2)} - b_i^{(k-1)}\right) \equiv \sum_{i=0}^{n_{k-1}} \omega_i^{(k)} \varphi_i^{(k-1)}(\mathbf{x}),$$

where the functions $\{\varphi_i^{(k-1)}(\mathbf{x})\}_{i=0}^{n_{k-1}}$ are given by

$$\varphi_0^{(k-1)}(\mathbf{x}) = 1 \quad \text{and} \quad \varphi_i^{(k-1)}(\mathbf{x}) = \sigma\left(\boldsymbol{\omega}_i^{(k-1)} \cdot \mathbf{x}^{(k-2)} - b_i^{(k-1)}\right).$$

Note that $C \in \mathcal{C}^{(k-1)}$ is either an isolated physical subdomain or consists of several connected physical sub-domains in $\hat{\mathcal{K}}^{(k-1)}$.

A heuristic method is introduced here to initialize a neuron such that its corresponding break poly-lines can divide the sub-domains in C as many as possible (please refer to Fig. 2 for a graphical illustration):

- step 0: Initialize an empty set $X = \emptyset$;

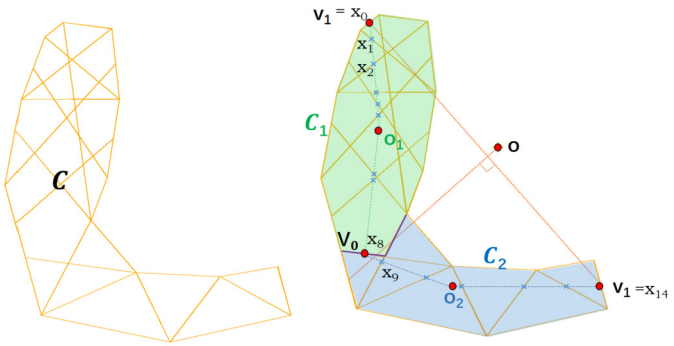


Fig. 2. A heuristic method for initializing a neuron at the last hidden layer.

- step 1: Compute pairwise distances among mid-points on the boundary edges of C, find the point pair $(\mathbf{v}_1, \mathbf{v}_2)$ having the longest distance;
- step 2: Compute the centroid $\mathbf{o}$ of C;
- step 3: If $\mathbf{o} \in C$, find the set of intersection points of the edges in C and the two line segments $\overline{\mathbf{o}\mathbf{v}_1}$ and $\overline{\mathbf{o}\mathbf{v}_2}$. Add the set of intersection points into X;
- step 4: If $\mathbf{o} \notin C$, decompose C into two sub-regions C_1 and C_2 by the line passing through $\mathbf{o}$ and perpendicular to $\overline{\mathbf{v}_1\mathbf{v}_2}$. Along the boundary edges of C_1 and C_2, locate a mid-point $\mathbf{v}_0$ which has the largest distance sum to $\mathbf{v}_1$ and $\mathbf{v}_2$;
- step 5: For each sub-region, use $(\mathbf{v}_0, \mathbf{v}_1)$ and $(\mathbf{v}_0, \mathbf{v}_2)$ respectively as the farthest point pair, repeat step 2-5 recursively until all sub-regions have their centroids located inside the region.

The above procedure returns a point set $X = \{\mathbf{x}_0, ..., \mathbf{x}_m\}$. Now, a reasonable initial is to choose $\{\omega_i^{(k)}\}_{i=0}^{n_{k-1}}$ so that the corresponding function $l_C(\mathbf{x})$ vanishes at $\mathbf{x}_j$ for all $0 \le j \le m$, i.e.,

$$0 = l_C(\mathbf{x}_j) = \sum_{i=0}^{n_{k-1}} \omega_i^{(k)} \varphi_i^{(k-1)}(\mathbf{x}_j) = \mathbf{l}_j^T \boldsymbol{\omega}^{(k)} \quad \text{for } j = 0, 1, ..., m, \tag{5.2}$$

where $\mathbf{x}_j^{(0)} = \mathbf{x}_j$, $\mathbf{x}_j^{(k-2)} = N^{(k-2)} \circ \cdots \circ N^{(1)}(\mathbf{x}_j)$ for $k > 2$, and $\mathbf{l}_j = \left(1, \varphi_1^{(k-1)}(\mathbf{x}_j), \cdots, \varphi_{n_{k-1}}^{(k-1)}(\mathbf{x}_j)\right)^T$. When $m < n_{k-2}$, any nontrivial solution of (5.2) may serve as an initial of $\boldsymbol{\omega}^{(k)}$. When $m \ge n_{k-2}$, (5.2) becomes an over-determined system and may not have a solution. In that case, we can choose a smaller γ_1 value in (4.3) so that fewer number of elements are marked.

When neurons are added to the current layer in the case (3), the initialization procedure described above needs to be changed as follows. Note that each $C \in \mathcal{C}^{(k-1)}$ may be identified as a subset of $\tilde{C}$ consisting of m connected physical sub-domains in $\mathcal{K}^{(k-2)}$. This implies that the $\{\mathbf{x}_j\}_{j=0}^m$ in (5.2) should be chosen based on the physical subdomains of $\tilde{C}$ in $\mathcal{K}^{(k-2)}$.

6. Numerical results for learning function

This section presents numerical results of the ANE method for learning a given function through the least-squares loss function. The test problem is a function defined on the domain $\Omega = [-1, 1]^2$ given by

$$f(x, y) = \tanh\left(\frac{1}{\alpha}(x^2 + y^2 - \frac{1}{4})\right) - \tanh\left(\frac{3}{4\alpha}\right), \tag{6.1}$$

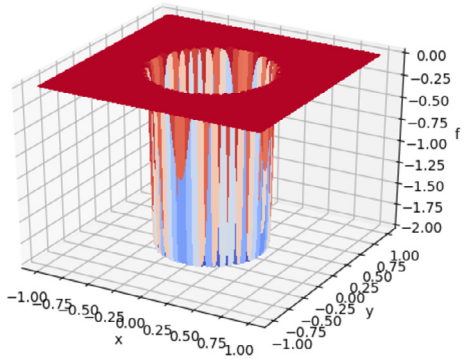
which exhibits a sharp transitional layer across a circular interface for small α. This test problem was used in [18] to motivate the ANE method for generating a multi-layer neural network. To learn $f(x, y)$ accurately, we show numerically that it is necessary to use at least a three-layer NN. The structure of a two- or three-layer NN is expressed as 2-n_1-1 or 2-n_1-n_2-1, respectively, where n_i is the number of neurons at the i^{th} hidden layer.

In this experiment, we set $\alpha = 0.01$ and the corresponding function f is depicted in Fig. 3 (a); a fixed 200×200 quadrature points are uniformly distributed in the domain Ω; we use the bulk marking strategy defined in (4.3) with $\gamma_1 = 0.5$; and we choose the expectation rate $\delta = 0.6$ with $r = 1$ in (4.4) and the tolerance $\epsilon = 0.05$. The ANE method starts with a two-layer NN of 12 neurons, and the corresponding breaking lines $\{\mathcal{P}_i\}_{i=1}^{12}$ are initialized uniformly. Specifically, half of breaking lines are parallel to the x-axis

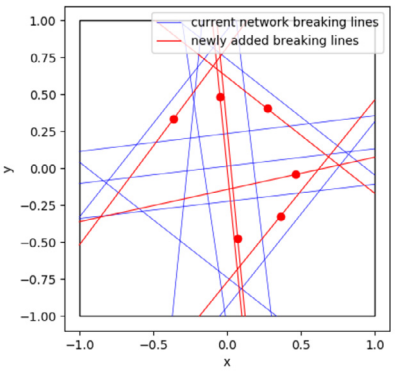
$$\boldsymbol{\omega}_i^{(1)} = 0 \quad \text{and} \quad b_i^{(1)} = -1 + \frac{1}{3}i \quad \text{for} \quad i = 0, \cdots, 5$$

Table 1
Adaptive numerical results for function with a transitional layer.

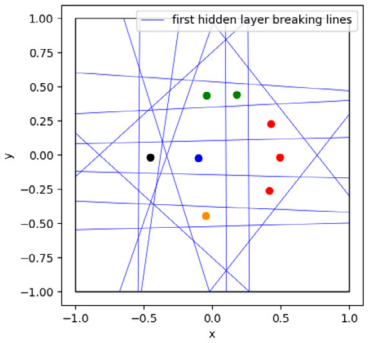
Network structure	# parameters	Training accuracy $\|f-\hat{f}\|_{\mathcal{T}}/\|f\|$	Improvement rate η
2-12-1	37	0.357414	–
2-18-1	**55**	**0.323118**	**0.293198**
2-26-1	93	0.272614	0.382528
2-18-5-1	**137**	**0.025483**	**1.538967**



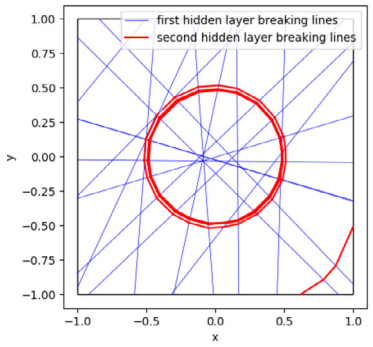
(a) The target function f with a circular transitional layer



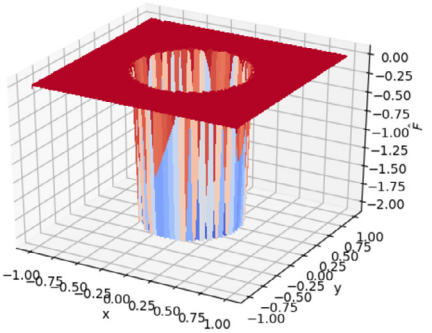
(b) PP of the approximation using 2-12-1 NN and centers of the marked elements (red dots)



(c) PP by 2-18-1 NN and isolated and connected sub-domains (dots)



(d) PP by adaptive 2-18-5-1 NN



(e) Approximation using adaptive 2-18-5-1 NN

Fig. 3. Adaptive approximation results for function with a transitional layer. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

and the other half are parallel to the y-axis

$$\boldsymbol{\omega}_i^{(1)} = \pi/2 \quad \text{and} \quad b_i^{(1)} = -1 + \frac{1}{3}(i-6) \quad \text{for} \quad i = 6, \cdots, 12.$$

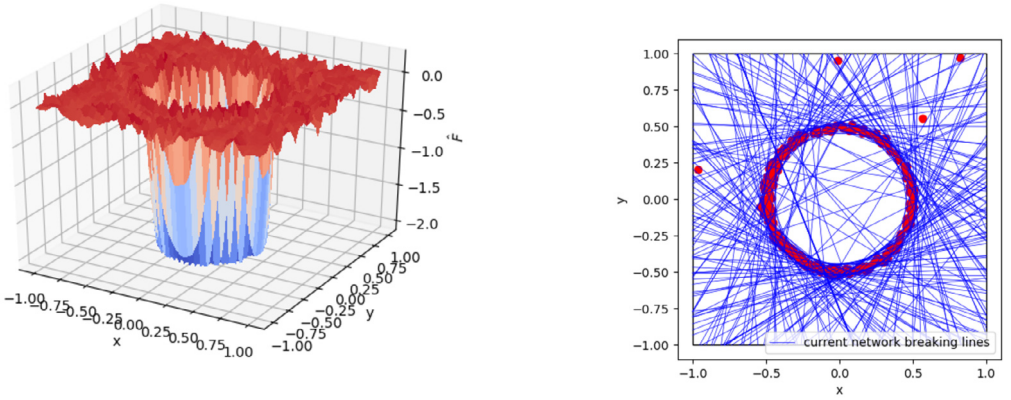
In addition, the output weights and bias are initialized by solving the linear system in (5.1).

For each iteration of the ANE method, the corresponding minimization problem in (2.6) is solved iteratively by the Adam version of gradient descent [15] with a fixed learning rate 0.005. The Adam's iterative solver is terminated when the relative change of the loss function $\|f-\hat{f}\|_{\mathcal{T}}$ is less than 10^{-3} during the last 2000 iterations.

The ANE process is automatically terminated after four loops (see Table 1), and the final model of NN generated by the ANE is 2-18-5-1 with 137 parameters. The best least-squares approximation of the final NN model and the corresponding physical partition are depicted in Figs. 3 (e) and (d). Clearly, the ANE method, using a relatively very small number of degrees of freedom, is capable of accurately approximating a function with thin layer without oscillation. This striking approximation property of the ANE method may be explained by the fact that the circular interface of the underlying function is captured very well by a couple of piece-wise breaking poly-lines of the approximation generated by *the second layer*.

Table 2
Numerical results of adaptive and fixed NNs for function with a transitional layer.

Network structure	# parameters	Training accuracy $\|f - \hat{f}\|_{\mathcal{T}}/\|f\|$
2-18-5-1 (Adaptive)	137	0.025483
2-18-5-1 (Fixed)	137	0.046199
2-174-1 (Fixed)	523	0.111223



(a) Approximation using fixed 2-174-1 NN (b) PP of the approximation by 2-174-1 NN and centers of elements with large errors (red)

Fig. 4. Approximation results generated by a fixed *2-174-1* NN for function with a transitional layer.

Figs. 3 (b)-(c) depict the physical partitions of the approximations at the intermediate NNs. In Fig. 3 (b), centers of the marked elements are illustrated by red dots; the breaking lines corresponding to the old and new neurons are displayed by blue and red lines, respectively. Table 1 shows that the adaptive network enhancement is done first at the current layer and then ended at the second hidden layer, because the improvement rates are smaller than the expectation rate for two consecutive network enhancement steps. Fig. 3 (c) shows that there are 8 marked sub-domains and 5 connected sub-domains, which explains only 5 neurons are added at the second hidden layer.

For the purpose of comparison, in Table 2 we also report numerical results produced by two fixed NN models. With the same architecture of NN, the first two rows of Table 2 imply that the adaptive NN obtains a better training result than the fixed NN. This suggests that the ANE method does provide a good initialization. The second experiment uses a fixed one hidden layer with nearly four times more parameters than the adaptive NN; its approximation is less accurate (see the third row of Table 2) and exhibits a certain level of oscillation (see Fig. 4 (a)) which is not acceptable in many applications. Despite that the corresponding physical partition (see Fig. 4(b)) does capture the circular interface, it is too dense in the region where the function does not have much fluctuation. This experiment indicates that a three layer NN is necessary for approximating a function with thin layer.

7. Application to PDEs

The ANE method introduced in this paper can be easily applied for learning solutions of partial differential equations. As an example, we demonstrate its application to the linear advection-reaction problem with discontinuous solution in this section.

7.1. Linear advection-reaction problem

Let Ω be a bounded domain in $\mathbb{R}^d$ with Lipschitz boundary, and $\boldsymbol{\beta}(\mathbf{x}) = (\beta_1, \cdots, \beta_d)^T \in C^1(\bar{\Omega})^d$ be the advective velocity field. Denote the inflow part of the boundary $\partial\Omega$ by

$$\Gamma_- = \{\mathbf{x} \in \Gamma : \boldsymbol{\beta}(\mathbf{x}) \cdot \boldsymbol{n}(\mathbf{x}) < 0\},$$

where $\boldsymbol{n}(\mathbf{x})$ is the unit outward normal vector to Γ_- at $\mathbf{x} \in \Gamma_-$. Consider the following linear advection-reaction equation

$$\begin{cases} u_{\boldsymbol{\beta}} + \gamma\, u = f & \text{in } \Omega, \\ u = g & \text{on } \Gamma_-, \end{cases} \tag{7.1}$$

where $u_{\boldsymbol{\beta}} = \boldsymbol{\beta}\cdot\nabla v$ is the directional derivative along the advective velocity field $\boldsymbol{\beta}$; and $\gamma \in C(\bar{\Omega})$, $f \in L^2(\Omega)$, and $g \in L^2(\Gamma_-)$ are given scalar-valued functions.

Introduce the solution space of (7.1) and the associated norm as follows

$$V_{\boldsymbol{\beta}} = \{v \in L^2(\Omega) : v_{\boldsymbol{\beta}} \in L^2(\Omega)\} \quad \text{and} \quad |||v|||_{\boldsymbol{\beta}} = \left(\|v\|_{0,\Omega}^2 + \|v_{\boldsymbol{\beta}}\|_{0,\Omega}^2\right)^{1/2},$$

respectively. Define the least-squares functional by

$$\mathcal{L}(v;\mathbf{f}) = \|v_{\boldsymbol{\beta}} + \gamma\, v - f\|_{0,\Omega}^2 + \|v - g\|_{-\boldsymbol{\beta}}^2 \tag{7.2}$$

for all $v \in V_{\boldsymbol{\beta}}$, where $\mathbf{f} = (f, g)$ and the weighted norm over the inflow boundary is defined by

$$\|v\|_{-\boldsymbol{\beta}} = \langle v, v\rangle_{-\boldsymbol{\beta}}^{1/2} = \left(\int_{\Gamma_-} |\boldsymbol{\beta}\cdot\boldsymbol{n}|\, v^2\, ds\right)^{1/2}.$$

Now, the least-squares formulation of (7.1) (see, e.g., [3,9]) is to find $u \in V_{\boldsymbol{\beta}}$ such that

$$\mathcal{L}(u;\mathbf{f}) = \min_{v\in V_{\boldsymbol{\beta}}} \mathcal{L}(v;\mathbf{f}). \tag{7.3}$$

Assume that there exist a positive constant γ_0 such that

$$\gamma(\mathbf{x}) - \frac{1}{2}\nabla\cdot\boldsymbol{\beta}(\mathbf{x}) \geq \gamma_0 > 0 \quad \text{for all } \mathbf{x}\in\Omega. \tag{7.4}$$

It then follows from the trace, triangle, and Poincaré inequalities that the homogeneous LS functional $\mathcal{L}(v;\mathbf{0})$ is equivalent to the norm $|||v|||_{\boldsymbol{\beta}}^2$, i.e., there exist positive constants α and M such that

$$\alpha\, |||v|||_{\boldsymbol{\beta}}^2 \leq \mathcal{L}(v;\mathbf{0}) \leq M\, |||v|||_{\boldsymbol{\beta}}^2. \tag{7.5}$$

7.2. LSNN method and a posteriori error estimator

Denote by $\mathcal{M}_N(\boldsymbol{\theta}, l)$ the set of DNN functions as in Section 2. Let $\mathcal{T}$ be a partition of the domain Ω and $\mathcal{E}_-$ as a partition of the inflow boundary Γ_-. Let $\mathbf{x}_K$ and $\mathbf{x}_E$ be the centroids of $K\in\mathcal{T}$ and $E\in\mathcal{E}_-$, respectively. Then the least-squares neural network (LSNN) method introduced in [17] is to find $u_{\mathcal{T}}^N(\mathbf{x},\boldsymbol{\theta}^*) \in \mathcal{M}_N(\boldsymbol{\theta}, l)$ such that

$$\mathcal{L}_{\mathcal{T}}\big(u_{\mathcal{T}}^N(\mathbf{x},\boldsymbol{\theta}^*);\mathbf{f}\big) = \min_{v\in\mathcal{M}_N(\boldsymbol{\theta},l)} \mathcal{L}_{\mathcal{T}}\big(v(\mathbf{x};\boldsymbol{\theta});\mathbf{f}\big) = \min_{\boldsymbol{\theta}\in\mathbb{R}^N} \mathcal{L}_{\mathcal{T}}\big(v(\mathbf{x};\boldsymbol{\theta});\mathbf{f}\big), \tag{7.6}$$

where the discrete LS functional is given by

$$\mathcal{L}_{\mathcal{T}}\big(v(\mathbf{x};\boldsymbol{\theta});\mathbf{f}\big) = \sum_{K\in\mathcal{T}} \big(v_{\boldsymbol{\beta}} + \gamma\, v - f\big)^2(\mathbf{x}_K;\boldsymbol{\theta})\,|K| + \sum_{E\in\mathcal{E}_-} \big(|\boldsymbol{\beta}\cdot\boldsymbol{n}|(v-g)^2\big)(\mathbf{x}_E;\boldsymbol{\theta})|E|.$$

Here, $|K|$ and $|E|$ are the d and $d-1$ dimensional measures of K and E, respectively.

There are two key components in applying the ANE method: (a) an *a posteriori* error estimator for determining if the current approximation is within the prescribed tolerance and (b) *a posteriori* error indicators for determining how many new neurons to be added at either width or depth. As a gift from the LS principle, the value of the least-squares functional at the current approximation is a good *a posteriori* error estimator. Specifically, let $u_k \in \mathcal{M}_N(\boldsymbol{\theta}, l)$ be the LSNN approximation at the current network and u be the exact solution of (7.1), then the estimator is given by

$$\xi \equiv \sqrt{\mathcal{L}_{\mathcal{T}}(u_k;\mathbf{f})} = \mathcal{L}_{\mathcal{T}}^{1/2}(u_k;\mathbf{f}). \tag{7.7}$$

To estimate the relative error, we may use

$$\xi_{\text{rel}} = \frac{\mathcal{L}_{\mathcal{T}}^{1/2}(u_k;\mathbf{f})}{\mathcal{L}_{\mathcal{T}}^{1/2}(u_k;\mathbf{0})}.$$

Lemma 7.1. *The estimator ξ satisfies the following reliability bound:*

$$|||u - u_k|||_{\boldsymbol{\beta}} \leq \frac{1}{\sqrt{\alpha}}\sqrt{\mathcal{L}(u_k;\mathbf{f})} \leq \frac{1}{\sqrt{\alpha}}\,\xi + h.o.t., \tag{7.8}$$

where h.o.t. means a higher order term.

Table 3
Adaptive numerical results for the solution with a constant jump over two line segments.

Network structure	# parameters	$\frac{\|u-\tilde{u}_\tau\|_0}{\|u\|_0}$	$\xi_{\text{rel}} = \frac{\mathcal{L}^{1/2}(\tilde{u}_\tau;\mathbf{f})}{\mathcal{L}^{1/2}(\tilde{u}_\tau;\mathbf{0})}$	Improvement rate η
2-6-1	19	0.543526	0.462477	–
2-7-1	**22**	**0.541213**	**0.449957**	**0.328366**
2-8-1	25	0.545274	0.449094	0.022159
2-7-1-1	24	0.515736	0.401161	2.120618
2-7-2-1	33	0.510399	0.391292	0.159051
2-7-3-1	42	0.113705	0.066804	4.510882
2-7-4-1	**51**	**0.105822**	**0.019171**	**5.197913**

Proof. The first inequality in (7.8) is a direct consequence of the lower bound in (7.5) and the fact that $\mathcal{L}(u_k;\mathbf{f}) = \mathcal{L}(u - u_k;\mathbf{0})$. The second inequality in (7.8) follows from the fact that $\mathcal{L}(u_k;\mathbf{f}) = \mathcal{L}_{\mathcal{T}}(u_k;\mathbf{f}) + \text{h.o.t.}$ This completes the proof of the lemma. □

To define the local error indicators, we make use of the physical partition $\mathcal{K}^{(l-1)} = \{K\}$ of the current approximation (see Section 3). For each $K \in \mathcal{K}^{(l-1)}$, the indicator ξ_K is defined by

$$\xi_K = \left(\| (u_k)_{\boldsymbol{\beta}} + \gamma\, u_k - f \|^2_{0,K} + \int_{\Gamma_- \cap \partial K} |\boldsymbol{\beta}\cdot\boldsymbol{n}|\, u_k^2\, ds \right)^{1/2}. \tag{7.9}$$

7.3. Numerical experiment

In this section, we report numerical results for two test problems: (1) constant jump over two line segments and (2) non-constant jump over a straight line. In [5], we showed theoretically that a NN with at least two hidden layers is needed in order to accurately approximate their solutions. The purpose of this section is to demonstrate the efficacy of the ANE method for generating a nearly minimal NN to learn solutions of PDEs.

In both experiments, the integration is evaluated on a uniform partition of the domain with 100×100 points. The prescribed expectation rate in (4.4) is set at $\delta = 0.6$ with $r = 1$. For the iterative solver, a fixed learning rate 0.003 and the same stopping criterion as that in Section 6 are used. Finally, the ANE method starts at a two-layer NN with initialization described in Section 6.

7.3.1. Constant jump over two line segments

The first test problem is the problem in (7.1) defined on $\Omega = (0,1)^2$ with $\gamma = f = 0$ and a piece-wise constant advection velocity field

$$\boldsymbol{\beta} = \begin{cases} (1-\sqrt{2},1)^T, & (x,y) \in \Upsilon_1 = \{(x,y) \in \Omega :\ y < x\}, \\ (-1,\sqrt{2}-1)^T, & (x,y) \in \Upsilon_2 = \{(x,y) \in \Omega :\ y \geq x\}. \end{cases}$$

Denote the inflow boundary and its subset by

$$\Gamma_- = \{(x,0) : x \in (0,1)\} \cup \{(1,0)\} \cup \{(1,y) :\ y \in (0,1)\} \quad \text{and} \quad \Gamma_-^1 = \{(x,0) : x \in (0,43/64)\},$$

respectively. For the inflow boundary condition

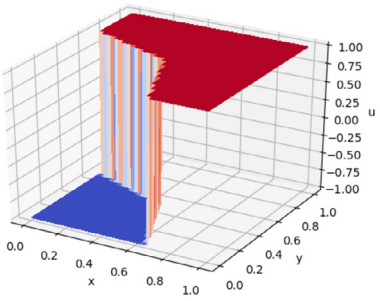
$$g(x,y) = \begin{cases} -1, & (x,y) \in \Gamma_-^1, \\ 1, & (x,y) \in \Gamma_-^2 = \Gamma_- \setminus \Gamma_-^1, \end{cases}$$

the exact solution of the problem is $u = -1$ in Ω_1 and $u = 1$ in $\Omega_2 = \Omega \setminus \bar{\Omega}_1$, where

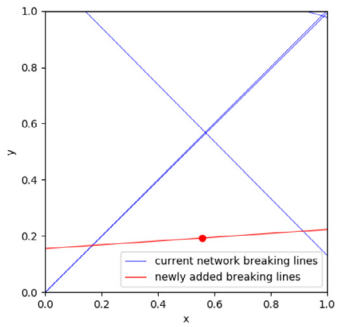
$$\Omega_1 = \cup_{i=1}^2 \{\mathbf{x} \in \Upsilon_i : \boldsymbol{\xi}_i \cdot \mathbf{x} < 43/64\},\ \boldsymbol{\xi}_1 = (1, \sqrt{2}-1)^T,\ \text{and}\ \boldsymbol{\xi}_2 = (\sqrt{2}-1, 1)^T.$$

As depicted in Fig. 5(a), the discontinuity of the solution is along two line segments.

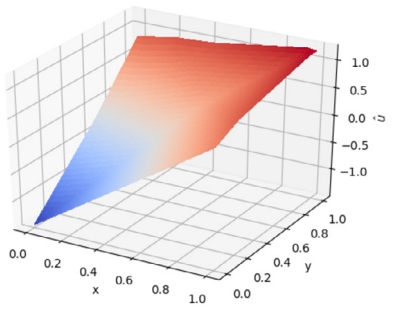
Choosing $\gamma_1 = 0.6$ for the bulk marking strategy in (4.3), the ANE method is terminated when the relative error estimator ξ_{rel} is less than the accuracy tolerance $\epsilon = 0.05$. The architecture of the final NN model of this test problem is 2-7-4-1 with $\xi_{\text{rel}} = 0.019171 < \epsilon = 0.05$ (see Table 3), and the corresponding approximation is depicted in Fig. 5 (f). Again, the corresponding physical partition (see Fig. 5 (e)) accurately captures the interface by the piece-wise breaking lines of the second hidden layer. This explains why the ANE method produces an accurate approximation to a discontinuous solution without oscillation or overshooting.



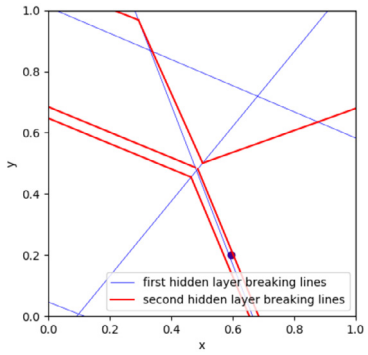
(a) Exact solution u



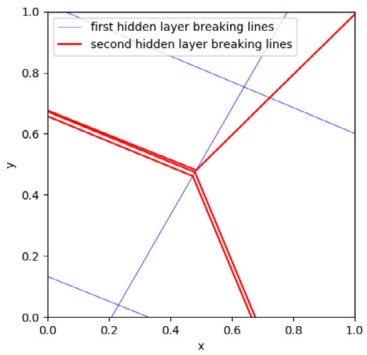
(b) PP by 2-6-1 NN, the marked element (red dot), and new breaking line (red line)



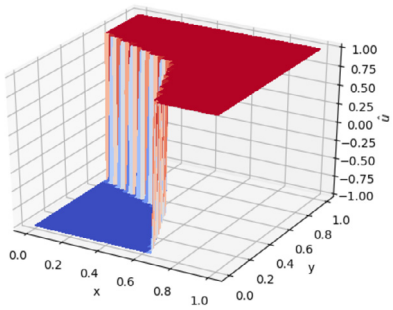
(c) Approximation by 2-7-1 NN



(d) PP by 2-7-3-1 NN and the marked element



(e) PP by adaptive 2-7-4-1 NN



(f) Approximation using adaptive 2-7-4-1 NN

Fig. 5. Adaptive approximation results for the solution with a constant jump over two line segments.

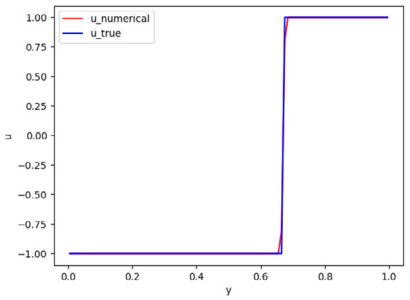
Table 4
Numerical results of adaptive and fixed NNs for the solution with a constant jump over two line segments.

Network structure	# parameters	$\frac{\|u-\bar{u}_\tau\|_0}{\|u\|_0}$	$\xi_{\text{rel}} = \frac{\mathcal{L}^{1/2}(\bar{u}_\tau;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_\tau;\mathbf{0})}$
2-7-4-1 (Adaptive)	51	0.105822	0.019171
2-7-4-1 (Fixed)	51	0.164322	0.116689

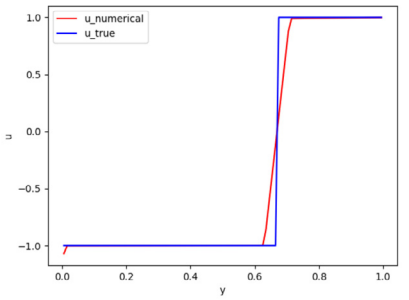
Approximation results of intermediate NNs are also reported in Table 3 and Fig. 5 (b)-(d). The second hidden layer is added when the improvement rate of two consecutive runs are less than the expectation rate (see the second and third rows in Table 3). Additionally, Fig. 5 (c) shows that a two-layer NN with seven neurons fails to approximate the discontinuous solution. This claim is actually true for a two-layer NN with 200 neurons (see [5]). Hence, a three-layer NN is essential for learning the solution of this problem.

A fixed 2-7-4-1 NN is tested for a comparison. Due to random initialization of some parameters, the experiment is replicated 10 times. We observe from the training process that this fixed network gets trapped easily at a local minimum and fails to approximate the solution well in most of the duplicate runs. The best result is reported in Table 4 and Fig. 6 (b). Although two network models have the same approximation power, attainable approximation may not be as accurate as the adaptive NN due to the inherent difficulty of non-convex optimization.

Remark 7.2. A fixed *2-5-5-1* NN was employed for the same test problem in *[5]*. Although a NN with fewer number of parameters can accurately approximate the solution, as pointed out in *Remark 5.1* in *[5]* that the network gets trapped easily at a local minimum. Repeated training is necessary for a fixed network model.

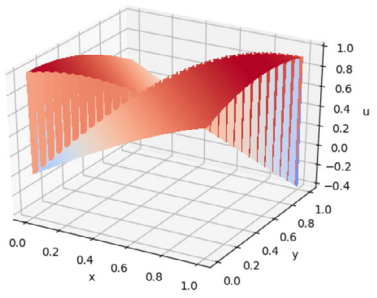


(a) Traces of the exact and numerical solutions on the plane $x = 0$ by adaptive 2-7-4-1 NN

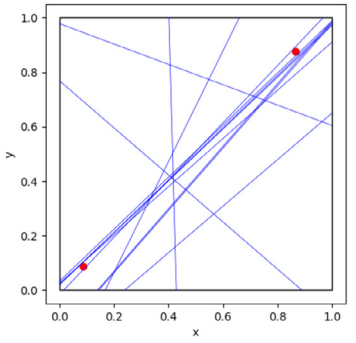


(b) Traces of the exact and numerical solutions on the plane $x = 0$ by fixed 2-7-4-1 NN

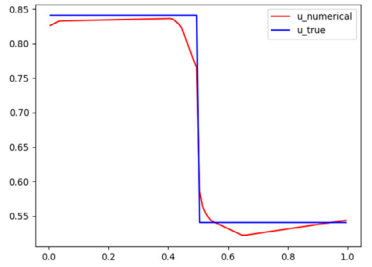
Fig. 6. Traces generated by adaptive and fixed NNs for the solution with a constant jump over two line segments.



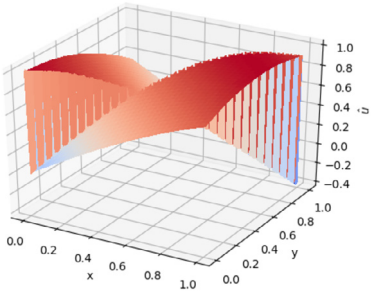
(a) Exact solution u



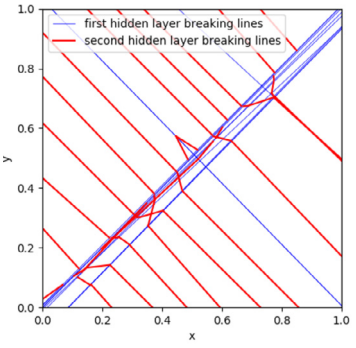
(b) PP by 2-13-1 NN and elements with large errors (red dots)



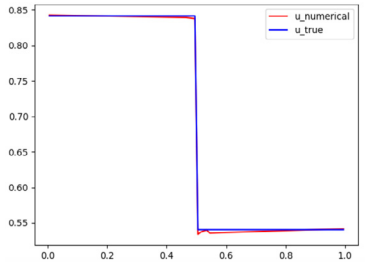
(c) Traces of exact and numerical solutions on $y = 1 - x$ by 2-13-1 NN



(d) Approximation by adaptive 2-13-10-1 NN



(e) PP by adaptive 2-13-10-1 NN



(f) Traces of exact and numerical solutions on $y = 1 - x$ by adaptive 2-13-10-1 NN

Fig. 7. Adaptive approximation results for the solution with a non-constant jump.

7.3.2. Non-constant jump over a straight line

The second test problem is again the equation in (7.1) defined on the domain $\Omega = (0, 1)^2$ with a constant advection velocity field and a piece-wise smooth inflow boundary condition. Specifically, $\gamma = 1$, $\boldsymbol{\beta} = (1, 1)^T/\sqrt{2}$, and $\Gamma_- = \Gamma_-^1 \cup \Gamma_-^2 \equiv \{(0, y) : y \in (0, 1)\} \cup \{(x, 0) : x \in (0, 1)\}$. Choose g and f accordingly such that the exact solution u is

$$u(x, y) = \begin{cases} \sin(x + y), & (x, y) \in \Omega_1 = \{(x, y) \in (0, 1)^2 : y > x\}, \\ \cos(x + y), & (x, y) \in \Omega_2 = \{(x, y) \in (0, 1)^2 : y < x\}. \end{cases}$$

As presented in Fig. 7 (a), the interface of the discontinuous solution is the diagonal line $y = x$ and the jump over the interface is not a constant.

Table 5
Adaptive numerical results for the solution with a non-constant jump.

Network structure	# parameters	$\frac{\|u-\tilde{u}_\tau\|_0}{\|u\|_0}$	$\xi_{\text{rel}}=\frac{\mathcal{L}^{1/2}(\tilde{u}_\tau;\mathbf{f})}{\mathcal{L}^{1/2}(\tilde{u}_\tau;\mathbf{0})}$	Improvement rate η
2-6-1	19	0.085907	0.178871	–
2-8-1	25	0.075888	0.157503	0.912494
2-10-1	31	0.070408	0.135401	1.340723
2-13-1	**40**	**0.070891**	**0.129806**	**0.365856**
2-15-1	46	0.068234	0.1250658	0.522031
2-13-2-1	57	0.042813	0.100613	1.290553
2-13-4-1	87	0.033823	0.091411	0.505859
2-13-7-1	132	0.029862	0.065525	1.429230
2-13-9-1	162	0.013429	0.044559	2.883692
2-13-10-1	**177**	**0.004651**	**0.025733**	**7.856341**

Table 6
Numerical results of adaptive and fixed NNs for the solution with a non-constant jump.

Network structure	# parameters	$\frac{\|u-\tilde{u}_\tau\|_0}{\|u\|_0}$	$\xi_{\text{rel}}=\frac{\mathcal{L}^{1/2}(\tilde{u}_\tau;\mathbf{f})}{\mathcal{L}^{1/2}(\tilde{u}_\tau;\mathbf{0})}$
2-13-10-1 (Adaptive)	177	0.004651	0.025733
2-13-10-1 (Fixed)	177	0.033602	0.049884

Starting at a two-layer NN with six neurons and choosing $\gamma_1 = 0.3$ in the bulk marking strategy in (4.3), the ANE process repeats itself multiple runs until the accuracy tolerance $\epsilon = 0.03$ is achieved. Ultimately, the ANE stops at a 2-13-10-1 NN model with the relative error estimator $\xi_{\text{rel}} = 0.025733$ (see Table 5). Fig. 7 (d) and (e) illustrate the approximation and the corresponding physical partition using the final model. In addition, the traces of the exact and numerical solutions on the plane $y = 1 - x$ are depicted in Fig. 7 (f), which clearly show that the final NN model is capable of accurately approximating the discontinuous solution without oscillation.

In Fig. 7 (b)-(c), we also present the traces of the exact and numerical solution and the corresponding physical partition using an intermediate 2-13-1 NN. Again, this two-layer NN fails to provide a good approximation (see Fig. 7 (c)) even though the corresponding physical partition (see Fig. 7(b)) locates the discontinuous interface. Moreover, as reported in Table 6, the adaptive model yields to a better approximation result comparing to a fixed ReLU NN model of the same size.

8. Conclusion

Designing an optimal deep neural network for a given task is important and challenging in many machine learning applications. To address this important, open question, we have proposed the adaptive network enhancement method for generating a nearly optimal multi-layer neural network for a given task within some prescribed accuracy. This self-adaptive algorithm is based on the novel network enhancement strategies introduced in this paper that determine when a new layer and how many new neurons should be added when the current NN is not sufficient for the given task. This adaptive algorithm learns not only from given information (data, function, PDE) but also from the current computer simulation, and it is therefore a learning algorithm at a level which is more advanced than common machine learning algorithms.

The resulting non-convex optimization at each adaptive step is computationally intensive and complicated with possible many global/local minimums. The ANE method provides a natural process for obtaining a good initialization that assists training significantly. Moreover, to provide a better initial guess, we have introduced an advanced procedure for initializing newly added neurons that are not at the first hidden layer.

In [18,17] and this paper, we have demonstrated that the ANE method can automatically design a nearly minimal two- or multi-layer NN to learn functions exhibiting sharp transitional layers as well as continuous/discontinuous solutions of PDEs. Functions and PDEs with sharp transitions or discontinuities at unknown location have been a computational challenge when approximated using other functional classes such as polynomials or piecewise polynomials with fixed meshes. In our future work, we plan to extend the applications of self-adaptive DNN to a broader set of tasks such as data fitting, classification, etc., where training data is limited but given. The ANE method has a potential to resolve the so-called "over-fitting" issue when data is noisy.

CRediT authorship contribution statement

Conception and design of study: Z. Cai, J. Chen, M. Liu.
Acquisition of data: J. Chen, M. Liu.
Analysis and/or interpretation of data: Z. Cai, J. Chen, M. Liu . Drafting the manuscript: Z. Cai, J. Chen, M. Liu.
Revising the manuscript critically for important intellectual content: Z. Cai, J. Chen, M. Liu.

Approval of the version of the manuscript to be published: Z. Cai, J. Chen, M. Liu.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E.L. Allgower, K. Georg, Numerical Continuation Methods: An Introduction, Springer-Verlag, Berlin, 1990.
[2] J. Berg, K. Nystrom, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[3] P. Bochev, J. Choi, Improved least-squares error estimates for scalar hyperbolic problems, Comput. Methods Appl. Math. 1 (2) (2001) 115–124.
[4] L. Bottou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2018) 223–311.
[5] Z. Cai, J. Chen, M. Liu, Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation, J. Comput. Phys. 443 (2021) 110514.
[6] Z. Cai, J. Chen, M. Liu, Least-squares ReLU neural network (LSNN) method for scalar nonlinear hyperbolic conservation law, Appl. Numer. Math. 174 (2022) 163–176, arXiv:2105.11627v1 [math.NA].
[7] Z. Cai, J. Chen, M. Liu, X. Liu, Deep least-squares methods: an unsupervised learning-based numerical method for solving elliptic PDEs, J. Comput. Phys. 420 (2020) 109707.
[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (2011) 2493–2537.
[9] H. De Sterck, T.A. Manteuffel, S.F. McCormick, L. Olson, Least-squares finite element methods and algebraic multigrid solvers for linear hyperbolic PDEs, SIAM J. Sci. Comput. 26 (1) (2004) 31–54.
[10] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 3.
[11] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey, J. Mach. Learn. Res. 20 (2019) 1–21.
[12] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[13] H. Guo, X. Zhuang, T. Rabczuka, Stochastic analysis of heterogeneous porous material with modified neural architecture search (NAS) based physics-informed neural networks using transfer learning, arXiv preprint arXiv:2010.12344v2 [cs.LG], 2020.
[14] D.O. Hebb, The Organization of Behavior. A Neuropsycholocigal Theory, A Wiley Book in Clinical Psychology, vol. 62, 1949, 78.
[15] D.P. Kingma, J. Ba, ADAM: a method for stochastic optimization, in: International Conference on Representation Learning, San Diego, 2015, arXiv preprint arXiv:1412.6980.
[16] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012, pp. 1097–1105.
[17] M. Liu, Z. Cai, Adaptive two-layer ReLu neural network: II. Ritz approximation to elliptic PDEs, Comput. Math. Appl. (2022), submitted for publication, arXiv:2107.08935v1 [math.NA].
[18] M. Liu, Z. Cai, J. Chen, Adaptive two-layer ReLU neural network: I. best least-squares approximation, Comput. Math. Appl. (2022), in press, arXiv: 2107.08935v1 [math.NA].
[19] A. Pinkus, Approximation theory of the mlp model in neural networks, Acta Numer. 8 (1999) 143–195.
[20] M. Raissia, P. Perdikarisb, G. Karniadakisa, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[21] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (1958) 386.
[22] L. Schumaker, Spline Functions: Basic Theory, Wiley, New York, 1981.
[23] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1139–1364.