Computers and Mathematics with Applications 113 (2022) 34–44

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

www.elsevier.com/locate/camwa

Adaptive two-layer ReLU neural network: I. Best least-squares approximation ☆

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

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

ARTICLE INFO

Keywords:
Adaptivity
Least-squares approximation
Neural network
ReLU activation

ABSTRACT

In this paper, we introduce adaptive network enhancement (ANE) method for the best least-squares approximation using two-layer ReLU neural networks (NNs). For a given function $f(\mathbf{x})$, the ANE method generates a two-layer ReLU NN and a numerical integration mesh such that the approximation accuracy is within the prescribed tolerance. The ANE method provides a natural process for obtaining a good initialization which is crucial for training nonlinear optimization problems. Numerical results for functions of two variables exhibiting either intersecting interface singularities or sharp interior layers demonstrate efficiency of the ANE method.

1. Introduction

Deep neural networks (DNNs) have achieved astonishing performance in computer vision, natural language processing, and many other artificial intelligence tasks. This success encourages wide applications to other fields, including recent studies of using DNN models to numerically solve partial differential equations (PDEs). Despite their great successes in many practical applications, it is widely accepted that approximation properties of DNNs are not yet well-understood and that understandings on why and how they work could lead to significant improvements. This explains rapidly increasing interests in theoretical and algorithmic studies of DNNs during recent years.

DNNs produce a new class of functions through compositions of linear transformations and activation functions. Their studies and applications may be traced back to the work of Hebb [1] in the late 1940's and Rosenblatt [2] in the 1950's. An often cited theoretical results on DNNs are the so-called universal approximation property [3,4], e.g., a two-layer NN is dense in $C(\Omega)$ for any compact subset $\Omega \in \mathbb{R}^d$ provided that the activation function is not a polynomial. Moreover, order of approximation for functions in the Sobolev space has been obtained for two-layer NNs using various activation functions [5]. For results on approximation theory of DNNs before 2000, see a survey article by Pinkus [6] and references therein.

Despite many efforts and much impressive progress made by numerical analysts, computational scientists, and practitioners, approximation properties of DNNs remain an active and open research field. Without complete understanding of approximation properties of DNNs, current methods on design of network structures are empirical. Tuning of depth and width is tedious, mainly from experimental results in ablation studies which typically require domain knowledge about the underlying problems. This leads to a fundamental, open question in machine learning: *given a target function/PDE, what is the minimal network model required, in terms of width, depth, and number of parameters, to approximate the function/solution within the prescribed accuracy?*

The purpose of this paper is to introduce and study adaptive network enhancement (ANE) methods for the best least-squares approximation to a target function by a two-layer ReLU NN, and, hence, to address this open problem partially. Specifically, for a given target function $f(\mathbf{x})$ and a given tolerance $\epsilon > 0$, the ANE method generates a two-layer ReLU neural network such that the approximation accuracy is within the prescribed tolerance. One of key components of the ANE method for the best least-squares approximation to a given function is the enhancement strategy which determines how many new neurons to be added, when the current approximation is not within the given accuracy. To address this issue, we introduce a global and a local network enhancement strategies. The global enhancement is based on a fixed convergence rate (see (5.3); and the local one is done through local error indicators collected on the physical subdomains plus a proper neuron initialization (detailed in section 5). The ANE method for solving elliptic PDEs is presented in the companion paper [7].

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

https://doi.org/10.1016/j.camwa.2022.03.005
Received 8 March 2021; Received in revised form 1 January 2022; Accepted 4 March 2022

Another important ingredient is the numerical integration mesh for evaluating the loss function. For many problems in machine learning, integral of the $L^2(\Omega)$ norm is often computed numerically by stochastic sampling approach, which in turn leads to *theoretical* convergence rate independent of the dimension. Other numerical integration methods that are independent of the dimension include quasi-Monte Carlo method [8] and the sparse grid method [9]. For simplicity, in this paper, we use adaptive numerical integration based on "mid-point" quadrature on either uniform or composite mesh. The composite mesh here means those meshes obtained from adaptive mesh refinement (AMR), where refinement of an element is done by subdividing it into small uniform elements. The AMR method presented in the paper is suitable for low dimensional problems and may be replaced by any adaptive integration procedure such as adaptive version of Monte Carlo, quasi-Monte Carlo, or sparse grid, etc. if a high dimensional problem is considered.

Theoretically, we show that the total approximation error is bounded by the approximation error of the NN plus the error of numerical integration (see Theorem 4.1) under the assumption of the Marcinkiewicz problem. This indicates that numerical integration should be chosen to ensure at least the accuracy of the current NN. For simple problems, one may simply use a fine uniform mesh which is able to capture all local behaviors of the integrand. For computationally intensive problems, one might need to use local AMR to generate a proper composite mesh. The stopping criterion for the AMR is based on if the mesh refinement of numerical integration improves the approximation accuracy (see Algorithm 5.2). With AMR for numerical integration, the ANE method defined in Algorithm 5.3 is able to generate a two-layer ReLU NN and a composite numerical integration mesh such that the approximation accuracy is within the prescribed tolerance.

The values of the parameters are trained by iteratively "solving" the non-convex optimization problem in (4.2). This high dimensional, non-convex optimization problem tends to be computationally intensive and complicated. Currently, it is often solved by iterative optimization methods such as gradient descent (GD), Stochastic GD, Adam, etc. (see, e.g., [10] for a review paper in 2018 and references therein). Usually nonlinear optimizations have many solutions, and the desired one is obtained only if we start from a close enough first approximation. The ANE method provides a natural process for obtaining a good initialization. Starting with a relatively small NN, the approximation of the previous NN is already a good approximation to the current NN in the loops of the ANE method. To provide a better approximation than the previous one, we divide all network parameters into two groups: linear parameters (output layer weights and bias) and nonlinear parameters (hidden layer weights and biases). Initialization of nonlinear parameters is based on their physical partitioning of the domain and initial of linear parameters are obtained by solving a system of linear equations with given nonlinear parameters.

The paper is organized as follows. Section 2 presents two-layer ReLU NNs. The best least-squares approximation and its discrete counterpart are described in sections 3 and 4, respectively. The ANE method is introduced in section 5, and initialization of parameters at different stage is proposed in section 6. Finally, numerical experiments for functions with intersecting interface singularities and interior layer like discontinuities are given in section 7, and conclusion in section 8.

2. Two-layer ReLU neural network

A two-layer NN consists of an input and output layers. The output layer does not have an activation function. Layers other than the output layer are called hidden layers. So a two-layer NN is also referred to as a one-hidden layer NN.

In d-dimension, for $i = 1, 2, ..., n$, let $\boldsymbol{\omega}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$ be the weights and bias of the first (input) layer, respectively; and let $c_i \in \mathbb{R}$ and $c_0 \in \mathbb{R}$ be the respective weights and bias of the second (output) layer. Then a two-layer ReLU NN with n neurons produces the following set of functions:

$$\hat{\mathcal{M}}_n(\sigma) = \left\{ c_0 + \sum_{i=1}^{n} c_i \sigma(\boldsymbol{\omega}_i \cdot \mathbf{x} - b_i) \ : \ c_i, b_i \in \mathbb{R}, \ \boldsymbol{\omega}_i \in \mathbb{R}^d \right\},$$

where σ is the rectified linear unit (ReLU) activation function given by

$$\sigma(t) = \max\{0, t\} = \begin{cases} 0, & t < 0, \\ t, & t \geq 0, \end{cases}$$

for any $t \in \mathbb{R}$. The $\sigma(t)$ is a continuous piece-wise linear function having a breaking point at $t = 0$ and belongs to a class of activation functions of the form

$$\sigma_k(t) = \big(\max\{0, t\}\big)^k = \begin{cases} 0, & t < 0, \\ t^k, & t \geq 0 \end{cases} \quad \text{for} \ \ k \in \mathbb{Z}_+,$$

where $\mathbb{Z}_+$ is the set of all positive integers. Note that $\sigma_k(t) \in C^{k-1}(\mathbb{R})$ is a piece-wise polynomial of degree k with a breaking point at $t = 0$. For simplicity of presentation, we restrict our attention to the ReLU activation function. Extension of results in this paper to general activation functions $\sigma_k(t)$ is straightforward.

There are $(d + 2)n + 1$ parameters for functions in the set $\hat{\mathcal{M}}_n(\sigma)$, where $n+1$ of them are the output weights and bias $\{c_i\}_{i=0}^n$ and $(d + 1)n$ of them are the input weights $\{\boldsymbol{\omega}_i\}_{i=1}^n$ and bias $\{b_i\}_{i=1}^n$. We refer to the former as linear parameters and the later nonlinear parameters. Thus, $\hat{\mathcal{M}}_n(\sigma)$ has of $n + 1$ linear and $(d + 1)n$ nonlinear parameters. To remove n nonlinear parameters, we notice that

$$\sigma(\boldsymbol{\omega} \cdot \mathbf{x} - b) = |\boldsymbol{\omega}| \, \sigma\left(\frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} \cdot \mathbf{x} - \frac{b}{|\boldsymbol{\omega}|} \right),$$

where $|\boldsymbol{\omega}| = \sqrt{\omega_1^2 + \cdots + \omega_d^2}$ is the length of a vector $\boldsymbol{\omega} \in \mathbb{R}^d$. This implies that $\hat{\mathcal{M}}_n(\sigma)$ is equal to

$$\mathcal{M}_n(\sigma, d) = \left\{ c_0 + \sum_{i=1}^{n} c_i \sigma(\boldsymbol{\omega}_i \cdot \mathbf{x} - b_i) \ : \ c_i, b_i \in \mathbb{R}, \ \boldsymbol{\omega}_i \in S^{d-1} \right\}, \tag{2.1}$$

where S^{d-1} is the unit sphere in $\mathbb{R}^d$. The number of parameters in $\mathcal{M}_n(\sigma, d)$ is

$$M(n, d) = (d + 1)n + 1.$$

Below let us look at $\mathcal{M}_n(\sigma, d)$ in one-, two- and d-dimension, separately. When $d = 1$, we have $S^0 = \{-1, 1\}$. Without loss of generality, we will choose $\omega_i = 1$ for all $i = 1, ..., n$. Then

$$\mathcal{M}_n(\sigma, 1) = \left\{ v(x, \theta) = c_0 + \sum_{i=1}^{n} c_i \sigma(x - b_i) \ : \ c_i, b_i \in \mathbb{R} \right\}, \tag{2.2}$$

where $\theta = (\mathbf{c}, \mathbf{b})$ denotes all parameters $\mathbf{c} = (c_0, c_1, ..., c_n)$ and $\mathbf{b} = (b_1, ..., b_n)$. The $\mathcal{M}_n(\sigma, 1)$ is the set of linear splines with n free knots that had been studied intensively in the late 1960s (see, e.g., [11]). It has been shown that the approximation of functions by linear splines can generally be dramatically improved if the knots are free [12]; particularly, the Gibbs phenomena for "rough" functions can be avoided [13].

In two dimensions ($d = 2$), S^1 is a unit circle:

$$\begin{aligned} S^1 &= \left\{ \boldsymbol{\omega} = (\omega_1, \omega_2)^t \in \mathbb{R}^2 \ : \ \omega_1^2 + \omega_2^2 = 1 \right\} \\ &= \left\{ \boldsymbol{\omega} = \big(\cos\gamma, \sin\gamma\big)^t \ : \ 0 \leq \gamma \leq 2\pi \right\}. \end{aligned}$$

This gives

$$\mathcal{M}_n(\sigma, 2) = \left\{ c_0 + \sum_{i=1}^{n} c_i \sigma\big((\cos\gamma_i) x_1 + (\sin\gamma_i) x_2 - b_i\big) \ : \ c_i, b_i \in \mathbb{R}, \right.$$
$$\left. \gamma_i \in [0, 2\pi] \right\}, \tag{2.3}$$

which is the set of continuous piece-wise linear functions with n free lines

$$l_i : (\cos\gamma_i)\,x_1 + (\sin\gamma_i)\,x_2 - b_i = 0 \quad \text{for } i = 1, ..., n. \tag{2.4}$$

Similarly, in the d-dimension, $\mathcal{M}_n(\sigma, d)$ is the set of continuous piecewise linear functions with n free hyper-planes

$$\mathcal{P}_i : \boldsymbol{\omega}_i \cdot \mathbf{x} - b_i = 0 \quad \text{for } i = 1, ..., n. \tag{2.5}$$

Clearly, $\mathcal{M}_n(\sigma, d)$ for $d \geq 2$ may be treated as a non-standard but beautiful extension of linear splines with free knots $\mathcal{M}_n(\sigma, 1)$ to multi-dimension.

Let

$$\varphi_i(\mathbf{x}) = \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i, b_i) = \sigma(\boldsymbol{\omega}_i \cdot \mathbf{x} - b_i),$$

which is a piece-wise linear function with free hyper-planes: $\boldsymbol{\omega}_i \cdot \mathbf{x} = b_i$ for $i = 1, ..., n$. Let $\varphi_0(\mathbf{x}) = \varphi_0(\mathbf{x}; \boldsymbol{\omega}_0, b_0) = 1$. For convenience of readers, we state and prove a well-known fact on the linear independence of $\{\varphi_i(\mathbf{x})\}_{i=0}^n$.

Lemma 2.1. *Assume that hyper-planes $\{\boldsymbol{\omega}_i \cdot \mathbf{x} = b_i\}_{i=1}^n$ are distinct. Then $\{\varphi_i(\mathbf{x}; \boldsymbol{\omega}_i, b_i)\}_{i=0}^n$ are linearly independent.*

Proof. Linear independence of $\varphi_0 = 1$ and $\varphi_1(\mathbf{x}; \boldsymbol{\omega}_1, b_1)$ is a direct consequence of the fact that $\varphi_1(\mathbf{x}; \boldsymbol{\omega}_1, b_1) \equiv 0$ on $\mathbb{R}^d \setminus \text{suppt}\{\varphi_1\}$. Assume that the lemma is valid for $n = k$, then linear independence of $\{\varphi_i(\mathbf{x}; \boldsymbol{\omega}_i, b_i)\}_{i=0}^{k+1}$ follows from the fact that $\sum_{i=0}^{k} c_i \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i, b_i) \equiv 0$ for all $\mathbf{x} \in \mathbb{R}^d \setminus \text{suppt}\{\varphi_{k+1}\}$ and the assumption that all hyper-planes $\{\boldsymbol{\omega}_i \cdot \mathbf{x} = b_i\}_{i=1}^{k+1}$ are distinct. This completes the proof of the lemma by induction. □

3. The best least-squares approximation

Denote vectors of weights and bias by

$$\mathbf{c} = (c_0, c_1, ..., c_n), \quad \boldsymbol{\omega} = (\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_n), \quad \text{and} \quad \mathbf{b} = (b_1, ..., b_n),$$

respectively, then each function $v \in \mathcal{M}_n(\sigma, d)$ may be represented as follows:

$$v(\mathbf{x}; \theta) = c_0 + \sum_{i=1}^{n} c_i\, \sigma(\boldsymbol{\omega}_i \cdot \mathbf{x} - b_i) = \sum_{i=0}^{n} c_i\, \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i, b_i), \tag{3.1}$$

where $\theta = (\mathbf{c}, \hat{\theta})$ with $\hat{\theta} = (\boldsymbol{\omega}, \mathbf{b})$ are parameters.

For a given function $f(\mathbf{x})$ defined on d-dimensional domain $\Omega \in \mathbb{R}^d$, the best least-squares approximation is to find $f_n(\mathbf{x}; \theta^*) \in \mathcal{M}_n(\sigma, d)$ such that

$$\|f(\cdot) - f_n(\cdot; \theta^*)\| = \min_{v \in \mathcal{M}_n(\sigma, d)} \|f - v\| = \min_{\theta \in \mathbb{R}^{M(n,d)}} \|f(\cdot) - v(\cdot; \theta)\|, \tag{3.2}$$

where $\|\cdot\|$ denotes the $L^2(\Omega)$ norm, $M(n, d)$ is the number of parameters defined in the previous section, and $v(\mathbf{x}; \theta)$ is given in (3.1). It was proven by Petrushev in [5] (see also [6]) that for any $f(\mathbf{x})$ in the Sobolev space $H^m(\Omega)$ for $m = 1, ..., 2 + \frac{d-1}{2}$, there exists a positive constant C such that

$$\|f - f_n\| \leq C\, n^{-m/d} \|f\|_{H^m(\Omega)}. \tag{3.3}$$

Remark 3.1. In one dimension, when $f \in L^p(\Omega)$ for $0 < p \leq \infty$, it was shown *(see Rice [14] and Powell [15])* that problem *(3.2)* has a solution $f_n \in C[0, 1]$. Solution of problem *(3.2)* is not unique in general; but it is unique for sufficiently smooth f and large enough n *(see Chui et al. [16])*.

Generally, $\mathcal{M}_n(\sigma, d)$ is only a set of functions. But for a fixed parameter $\hat{\theta}^0 = (\boldsymbol{\omega}^0, \mathbf{b}^0)$, the set $\mathcal{M}_n(\sigma, d)$ becomes a subspace

$$\mathcal{M}_n(\sigma, d) = \text{span}\,\{\varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0)\}_{i=0}^n.$$

Then the best least-squares approximation in (3.2) becomes to find $f_n^0 = \sum_{i=0}^{n} c_i^0\, \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0) \in \mathcal{M}_n(\sigma, d)$ such that

$$(f_n^0, \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0)) = (f, \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0)) \quad \forall\, i = 0, 1, ..., n,$$

where $(f, g) = \int_\Omega f(\mathbf{x}) g(\mathbf{x})\, d\mathbf{x}$ denotes the $L^2(\Omega)$ inner product. The corresponding system of algebraic equations is

$$\mathbf{M}(\hat{\theta}^0)\, \mathbf{c}^0 = F(\hat{\theta}^0), \tag{3.4}$$

where $\mathbf{M}(\hat{\theta}^0) = (M_{ij})_{(n+1)\times(n+1)}$ is the mass matrix with $M_{ij} = (\varphi_j(\mathbf{x}; \boldsymbol{\omega}_j^0, b_j^0), \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0))$, $\mathbf{c}^0 = (c_0^0, c_1^0, ..., c_n^0)^t$, and $F(\hat{\theta}^0) = (F_i)_{(n+1)\times 1}$ is the right-hand side vector with $F_i = (f, \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0))$.

Lemma 3.2. *Assume that the hyper-planes $\{\boldsymbol{\omega}_i^0 \cdot \mathbf{x} = b_i^0\}_{i=1}^n$ are distinct. Then the mass matrix $\mathbf{M}(\hat{\theta}^0)$ is symmetric, and positive definite.*

Proof. Clearly, $\mathbf{M}(\hat{\theta}^0)$ is symmetric. For any $\mathbf{v} = (v_0, v_1, ..., v_n)^t$, we have

$$\mathbf{v}^t \mathbf{M}(\hat{\theta}^0) \mathbf{v} = \|v\|^2,$$

where $v(\mathbf{x}) = \sum_{i=0}^{n} v_i \varphi_i(\mathbf{x}; \boldsymbol{\omega}_i^0, b_i^0)$. By Lemma 2.1, $\|v\|^2$ is positive for any nonzero $\mathbf{v}$, which, in turn, implies that $\mathbf{M}(\hat{\theta}^0)$ is positive definite. □

4. Effect of numerical integration

In practice, integral of the loss function is often computed numerically. A common practice in machine learning (see, e.g., [17–19]) uses Monte Carlo integration of the form

$$\mathcal{I}(v) = \int_\Omega v(\mathbf{x})\, d\mathbf{x} \approx \frac{|\Omega|}{N} \sum_{i=1}^{N} v(\mathbf{x}_i), \tag{4.1}$$

where $|\Omega|$ is the volume of the domain Ω and $\{\mathbf{x}_i\}_{i=1}^N$ are the sampling points randomly generated based on an assumed distribution of $\mathbf{x}$. This stochastic approach is simple and valid for any dimensions. Moreover, it leads to theoretical convergence rate independent of the dimension. Other numerical integration methods that are independent of the dimension include quasi-Monte Carlo method [8] and the sparse grid method [9].

In this paper, we use adaptive numerical integration as in [20] in line with the ANE method. For simplicity of presentation, we consider only "mid-point" quadrature on either uniform or composite mesh. The composite mesh here means those meshes obtained from adaptive mesh refinement (AMR), where refinement of an element is done by subdividing it into small uniform elements. To this end, let

$$\mathcal{T} = \{K : K \text{ is an open subdomain of } \Omega\}$$

be a partition of the domain Ω. Here, the partition means that 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; more precisely,

$$\bar{\Omega} = \cup_{K \in \mathcal{T}} \bar{K} \quad \text{and} \quad K \cap T = \emptyset, \quad \forall\, K, T \in \mathcal{T}.$$

Let $\mathbf{x}_T$ be the centroid of $T \in \mathcal{T}$. The $\mathbf{x}_T$ will be used as quadrature points which are fundamentally different from sampling points used in the setting of standard supervised learning. The composite "mid-point" quadrature rule is given by

$$\mathcal{I}(v) = \int_\Omega v(\mathbf{x})\, d\mathbf{x} \approx \sum_{T \in \mathcal{T}} v(\mathbf{x}_T)\, |T| \equiv \mathcal{Q}_{\mathcal{T}}(v),$$

where $|T|$ is the volume of element $T \in \mathcal{T}$. Similarly, one may use any quadrature rule such as composite trapezoidal, Simpson, Gaussian, etc.

Let $Q_{\mathcal{T}}$ be a quadrature operator, i.e., $\mathcal{I}(v) \approx Q_{\mathcal{T}}(v)$, such that

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

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

$$\|f(\cdot) - f_{\mathcal{T}}(\cdot;\theta_{\mathcal{T}}^*)\|_{\mathcal{T}} = \min_{v\in\mathcal{M}_n(\sigma,d)} \|f-v\|_{\mathcal{T}} = \min_{\theta\in\mathbb{R}^{M(n,d)}} \|f(\cdot) - v(\cdot;\theta)\|_{\mathcal{T}}, \quad (4.2)$$

Theorem 4.1. *Assume that there exists a positive constant α such that $\alpha\|v\|^2 \le \|v\|_{\mathcal{T}}^2$ for all $v \in \mathcal{M}_{2n}^1(\sigma,d)$. Let $f_{\mathcal{T}}$ be a solution of (4.2). Then there exists a positive constant C such that*

$$C\|f - f_{\mathcal{T}}\| \le \inf_{v\in\mathcal{M}_{2n}^1(\sigma,d)} \left\{ \|f-v\| + \sup_{w\in\mathcal{M}_{2n}^1(\sigma,d)} \frac{|(\mathcal{I}-Q_{\mathcal{T}})(vw)|}{\|w\|} \right\} + \sup_{w\in\mathcal{M}_{2n}^1(\sigma,d)} \frac{|(\mathcal{I}-Q_{\mathcal{T}})(fw)|}{\|w\|}. \quad (4.3)$$

Proof. Since $\mathcal{M}_n(\sigma,d)$ is a set, $f_{\mathcal{T}} \in \mathcal{M}_n(\sigma,d)$ is then characterized by the inequality

$$(f - f_{\mathcal{T}}, v - f_{\mathcal{T}})_{\mathcal{T}} \le 0 \quad \forall\, v \in \mathcal{M}_n(\sigma,d). \quad (4.4)$$

For any $v \in \mathcal{M}_n(\sigma,d)$, it follows from the assumption and (4.4) that

$$\begin{aligned}\alpha\|f_{\mathcal{T}} - v\|^2 &\le \|f_{\mathcal{T}} - v\|_{\mathcal{T}}^2 \le (f, f_{\mathcal{T}} - v)_{\mathcal{T}} - (v, f_{\mathcal{T}} - v)_{\mathcal{T}} \\ &= \Big((f, f_{\mathcal{T}} - v)_{\mathcal{T}} - (f, f_{\mathcal{T}} - v)\Big) + \Big((v, f_{\mathcal{T}} - v) - (v, f_{\mathcal{T}} - v)_{\mathcal{T}}\Big) \\ &\quad + (f - v, f_{\mathcal{T}} - v)\end{aligned}$$

which, together with the Cauchy-Schwarz inequality, implies

$$\alpha\|f_{\mathcal{T}} - v\| \le \sup_{w\in\mathcal{M}_{2n}^1(\sigma,d)} \frac{|(\mathcal{I}-Q_{\mathcal{T}})(fw)|}{\|w\|} + \sup_{w\in\mathcal{M}_{2n}^1(\sigma,d)} \frac{|(\mathcal{I}-Q_{\mathcal{T}})(vw)|}{\|w\|} + \|f - v\|.$$

Combining the above inequality with the triangle inequality

$$\|f - f_{\mathcal{T}}\| \le \|f - v\| + \|v - f_{\mathcal{T}}\|$$

and taking the infimum over all $v \in \mathcal{M}_{2n}^1(\sigma,d)$ yield (4.3). This completes the proof of the theorem. □

Theorem 4.1 indicates that the total error of the best least-squares approximation with numerical integration is bounded by the approximation error of the neural network and the error of the numerical integration. To ensure the approximation accuracy of the given neural network, we need to choose a numerical integration with a compatible accuracy, e.g., the composite "mid-point" numerical integration on an adaptively refined uniform partition.

Remark 4.2. The assumption in *Theorem 4.1* is known as the Marcinkiewicz problem in literature and has not been verified for functions in $\mathcal{M}_{2n}^1(\sigma,d)$. Recently, Temlyakov *[21]* introduced a new technique to systematically study this and related issues for functions in various finite dimensional subspaces.

5. Adaptive network enhancement (ANE) method

For a given target function $f(\mathbf{x})$, let $f_{\mathcal{T}}(\mathbf{x},\theta_{\mathcal{T}}^*)$ be the solution of problem (4.2). For a given tolerance $\epsilon > 0$, this section studies self-adaptive method for creating a two-layer ReLU NN and a numerical integration mesh such that the approximation accuracy is within the prescribed tolerance, i.e.,

$$\|f - f_{\mathcal{T}}\| \le \epsilon\|f\|. \quad (5.1)$$

First, we consider the case that the numerical integration based on a partition $\mathcal{T}$ is sufficiently accurate. Similar to the idea of the standard adaptive mesh-based numerical methods, we start with a two-layer ReLU NN with a small number of neurons, solve the optimization problem in (4.2), and estimate the total error by computing *a posteriori* error estimator

$$\xi = \|f - f_{\mathcal{T}}\|_{\mathcal{T}}/\|f\|. \quad (5.2)$$

If $\xi > \epsilon$, we then enhance the NN by adding new neurons and this procedure repeats until (5.1) is met. This process is referred as the adaptive network enhancement (ANE) and it generates a two-layer ReLU NN whose approximation to f satisfies a given approximation accuracy target.

An immediate key question for the ANE method is: how many new neurons will be added at each adaptive step? To address this issue, we propose two network enhancement strategies. One is global and the other is local. The global one is based on the assumption that the network approximation to the target function f has a fixed convergence rate α:

$$\hat{\xi}^{(k)} = \|f - f_{\mathcal{T}}^{(k)}\|_{\mathcal{T}} = \mathcal{O}(n_k^{-\alpha}),$$

where $f_{\mathcal{T}}^{(k)}$ is the approximation in $\mathcal{M}_{n_k}(\sigma,d)$, n_k is the number of neurons of the k^{th} NN, and α is the order of approximation. A simple calculation suggests the following number of neurons for the next network:

$$n_k = \min\left\{2n_{k-1}, \left\lceil \left(\hat{\xi}^{(k-1)}/\epsilon\right)^{1/\alpha_k} n_{k-1} \right\rceil\right\}, \quad (5.3)$$

where α_k is an approximation to the order α. For $k \ge 3$, $\alpha_k = \ln\left(\hat{\xi}^{(k-2)}/\hat{\xi}^{(k-1)}\right) \Big/ \ln\left(n_{k-1}/n_{k-2}\right)$. Possible choice for α_2 is 1 (linear rate) or some positive real number based on some *a priori* information of the target function.

To introduce our local network enhancement strategy, we notice that $\mathcal{M}_n(\sigma,d)$ is the set of continuous piece-wise linear functions with n free hyper-planes given by (2.5). For any bounded domain $\Omega \in \mathbb{R}^d$, these n hyper-planes plus the boundary of the domain Ω form a partition, $\mathcal{K}_n = \{K\}$, of the domain Ω. Again, the partition means that union of all subdomains of $\mathcal{K}_n$ equals the whole domain Ω and that any two distinct subdomains of $\mathcal{K}_n$ have no intersection. We will refer to $\mathcal{K}_n = \{K\}$ as the physical partition of the domain Ω.

This observation implies that the network enhancement strategy could make use of local errors on elements of the physical partition $\mathcal{K}_n$. Specifically, let us introduce local error indicator ξ_K for each element $K \in \mathcal{K}_n$:

$$\xi_K = \|f - f_{\mathcal{T}}\|_{K,\mathcal{T}} \equiv \left(\sum_{x_{K'}\in K} (f - f_{\mathcal{T}})^2(\mathbf{x}_{K'})|K'| \right)^{1/2}. \quad (5.4)$$

We then define a subset $\hat{\mathcal{K}}_n$ of $\mathcal{K}_n$ by using either the following average marking strategy:

$$\hat{\mathcal{K}}_n = \left\{ K \in \mathcal{K}_n : \xi_K \ge \frac{1}{\#\mathcal{K}_n} \sum_{K\in\mathcal{K}_n} \xi_K \right\}, \quad (5.5)$$

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

$$\sum_{K\in\hat{\mathcal{K}}_n}\xi_K^2 \geq \gamma_1 \sum_{K\in\mathcal{K}_n}\xi_K^2 \quad \text{for } \gamma_1\in(0,1). \tag{5.6}$$

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

With an accurate numerical integration, the ANE method is defined in Algorithm 5.1.

Algorithm 5.1 Adaptive two-layer ReLU 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 (4.2);
(2) estimate the total error by computing $\xi=\left(\sum_{K\in\mathcal{K}}\xi_K^2\right)^{1/2}/\|f\|_{\mathcal{T}}$, where $\mathcal{K}$ is the physical partition of the current approximation;
(3) if $\xi<\epsilon$, then stop; otherwise, go to Step (4);
(4) add new neurons to the network by using the network enhancement strategy, then go to Step (1).

Next, we consider adaptive mesh refinement (AMR) on numerical integration for a fixed NN. Let $f_{\mathcal{T}}(\mathbf{x},\theta_{\mathcal{T}}^*)$ be the solution of problem (4.2) associated with the partition $\mathcal{T}$. Let $\hat{\mathcal{T}}$ be a subset of $\mathcal{T}$ generated by using either the average or the bulk marking strategy. For each marked element $T\in\hat{\mathcal{T}}$, this d-dimensional cube is subdivided into 2^d small cubes of equal size. The new partition $\mathcal{T}'$ consists of elements in $\mathcal{T}\setminus\hat{\mathcal{T}}$ and new elements generated from $\hat{\mathcal{T}}$. Denote by $f_{\mathcal{T}'}(\mathbf{x},\theta_{\mathcal{T}'}^*)$ the solution of problem (4.2) associated with the partition $\mathcal{T}'$. For both solutions $f_{\mathcal{T}}$ and $f_{\mathcal{T}'}$ based on the mesh $\mathcal{T}$ and its refinement $\mathcal{T}'$, define the following global estimators:

$$\eta(f_{\mathcal{T}})=\left(\sum_{T\in\mathcal{T}}\eta_T(f_{\mathcal{T}})^2\right)^{1/2} \quad\text{and}\quad \eta(f_{\mathcal{T}'})=\left(\sum_{T\in\mathcal{T}'}\eta_T(f_{\mathcal{T}'})\right)^{1/2}.$$

where local indicators on $\mathcal{T}'$ are given by

$$\eta_T(f_{\mathcal{T}})=\|f-f_{\mathcal{T}}\|_{T,\mathcal{T}} \quad\text{and}\quad \eta_T(f_{\mathcal{T}'})=\|f-f_{\mathcal{T}'}\|_{T,\mathcal{T}'}. \tag{5.7}$$

The following algorithm generates a numerical integration mesh which ensures approximation accuracy of a given NN.

Algorithm 5.2 Adaptive Mesh Refinement with a fixed NN.

Given a target function $f(\mathbf{x})$ and the solution of problem (4.2) on the partition $\mathcal{T}$,

(1) refine $\mathcal{T}$ by the refinement strategy to obtain a new partition $\mathcal{T}'$
(2) solve the minimization problem in (4.2) on $\mathcal{T}'$;
(3) if $\eta(f_{\mathcal{T}'})\leq\gamma_2\eta(f_{\mathcal{T}})$, then go to Step (1) with $\mathcal{T}=\mathcal{T}'$; otherwise, output $\mathcal{T}$.

The stopping criterion used in Algorithm 5.2 is based on whether or not the mesh refinement on numerical integration improves approximation accuracy. When the refinement does not improve accuracy much, the AMR stops and outputs the current mesh.

Finally, we are ready to present adaptive network enhancement (ANE) method for a two-layer ReLU NN including AMR for numerical integration in Algorithm 5.3. The purpose of the AMR for numerical integration is to ensure approximation accuracy with less quadrature points than a fine uniform partition. Comparing with the ANE, the AMR is secondary.

6. Strategies for training (iterative solvers)

The exceptional power of DNNs in approximation comes with a price: the procedure for determining the values of the parameters is now a problem in nonlinear optimization. This high dimensional, nonlinear optimization problem tends to be computationally intensive and complicated. Currently, it is often solved by iterative optimization methods

Algorithm 5.3 Adaptive two-layer ReLU NN.

Given a target function $f(\mathbf{x})$ and a tolerance $\epsilon>0$, starting with a coarse uniform partition $\mathcal{T}_0$ of the domain Ω for numerical integration and with a two-layer ReLU NN with a small number of neurons,

(1) solve the minimization problem in (4.2);
(2) use Algorithm 5.2 to generate a numerical integration mesh $\mathcal{T}$;
(3) solve the minimization problem in (4.2) associated with $\mathcal{T}$;
(4) estimate the total error by computing $\xi=\left(\sum_{K\in\mathcal{K}}\xi_K^2\right)^{1/2}/\|f\|_{\mathcal{T}}$, where $\mathcal{K}$ is the physical partition of the current approximation;
(5) if $\xi<\epsilon$, then stop; otherwise, go to Step (6);
(6) add new neurons to the network by using the network enhancement strategy, then go to Step (1).

such as gradient descent (GD), Stochastic GD, Adam, etc. (see, e.g., [10] for a review paper in 2018 and references therein). Usually nonlinear optimizations have many solutions, and the desired one is obtained only if we start from a close enough first approximation. The ANE method provides a natural process for obtaining a good initialization. This section describes our initialization for all three stages of the ANE method.

The first stage is the beginning of the ANE method, in which we specify the size of the NN, both input and output weights and bias, and a partition of the domain for numerical integration. Due to the fact that input weights and bias determine physical locations of breaking hyper-planes, we first subdivide the domain Ω by a coarse, uniform partition and then distribute those breaking hyper-planes on the mesh of this partition. For example, when $\Omega=(0,1)^2$, the two-layer NN with $2(m_0+1)$ neurons use the following initial breaking lines:

$$x=\frac{i}{m_0} \quad\text{and}\quad y=\frac{i}{m_0} \quad \text{for } i=0,1,\ldots,m_0.$$

This breaking lines imply the following input weights and bias:

$$\theta_{1,i}=\left((1,0),\frac{i}{m_0}\right) \quad\text{and}\quad \theta_{2,i}=\left((0,1),\frac{i}{m_0}\right)$$

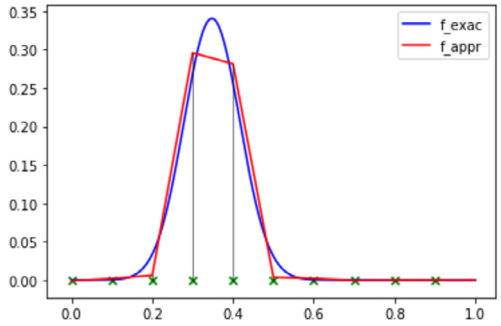
for $i=0,1,\ldots,m_0$. For numerical integration, we again start with a uniform partition $\mathcal{T}$ of the domain Ω which, in general, is much finer than the previous physical partition initializing the NN. Initial of the output weights and bias is given by the solution of the system of linear equations in (3.4).

The second stage is the AMR for numerical integration. For each new partition $\mathcal{T}$, natural initial of parameters θ is the corresponding values of the current approximation since the NN remains unchanged.

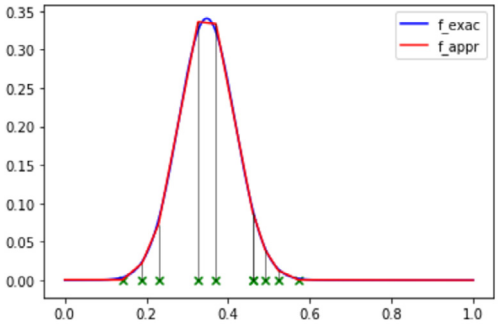
The third stage is when the NN is enhanced by adding new neurons. Clearly, parameters corresponding to old neurons will use the current approximation as their initial. To initialize corresponding parameters of new neurons, for the global enhancement strategy, one can add new neurons randomly; or add new neurons uniformly across the domain (i.e. set their input weights and biases with corresponding break hyper planes uniformly subdividing the domain). For the local enhancement strategy, we propose to make use of the subset $\hat{\mathcal{K}}_n$ marked in (5.5) or (5.6). For each element $K\in\hat{\mathcal{K}}_n$, we add one neuron whose initial is corresponding to the breaking hyper-plane that passes through the centroid of K and orthogonal to the direction vector with the smallest variance of quadrature points in K. This direction vector may be computed by the Principal Component Analysis method (or PCA [22]). For output weights and biases corresponding to new neurons, a simple initial is to set them zero. This means that the initial of the approximation is the current approximation. A better way is to solve problem (3.4) for all output weights and bias by using the current breaking hyper-planes for the input weights and bias.

7. Numerical experiments

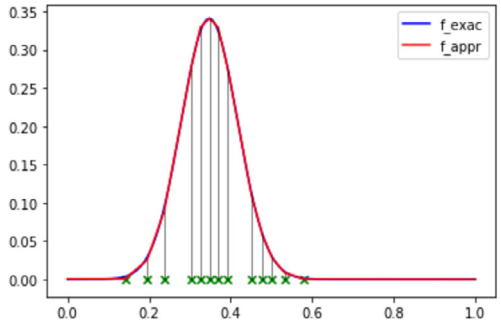
In this section, we present our numerical experiments on using ANE to approximate various functions. In all experiments, the minimization



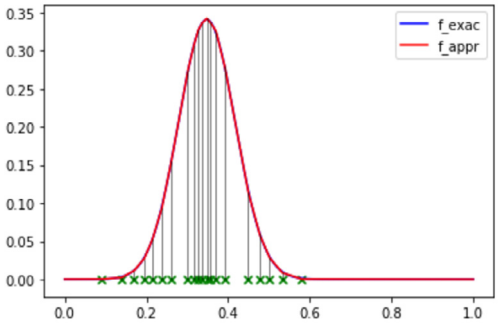
(a) Initial NN model with 10 uniform break points



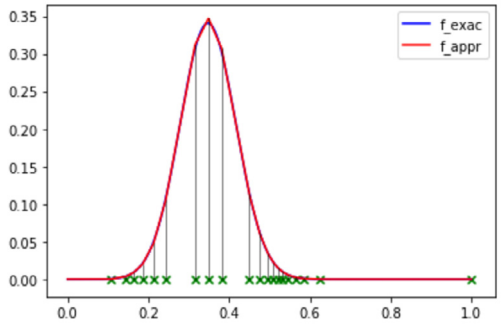
(b) Optimized NN model with 10 neurons



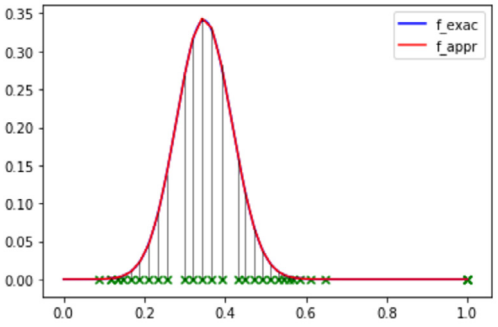
(c) Optimized NN model with 13 neurons using ANE



(d) Optimized NN model with 20 neurons using ANE



(e) Fixed NN model with 20 neurons



(f) Fixed NN model with 38 neurons

Fig. 1. Results of using two-layer ReLU networks for approximating function (7.1).

problem (4.2) is solved using the Adam version of gradient descent [23]. For each run during the adaptive process, the stopping criteria for the iterative solver is set as follows: the solver stops when the loss function $\|f - \hat{f}\|_{\mathcal{T}}$ decreases within 0.1% in the last 2000 iterations. This stopping criteria is set to explore the network approximation power without constraining the number of iterations.

7.1. Smooth function

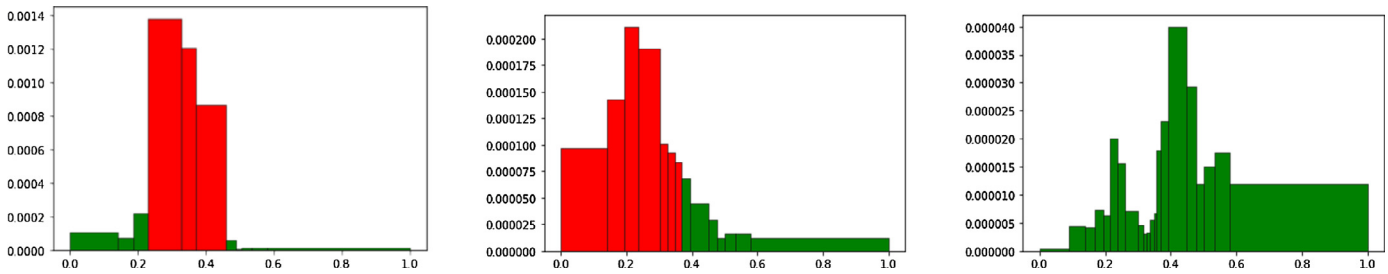
The first test problem is a smooth function of one variable

$$f(x) = x\left(e^{-(x-\frac{1}{3})^2/k} - e^{-\frac{4}{9}/k}\right), \tag{7.1}$$

which is defined on the interval $\Omega = [0, 1]$. When $k = 0.01$, this function is the solution to a Poisson equation studied in [24,20]. We use this simple toy problem to test the efficacy of the proposed ANE method.

The target approximation accuracy is set as $\epsilon = 0.005$. A fixed uniform partition $\mathcal{T}$ with 1000 quadrature points is used for this experiment. We start from 10 neurons for the input layer with their break points initialized uniformly across the domain, i.e., $b_i = 0.1i$ for $i = 0, 1, \cdots, 9$. The initial network model's output weights and biases are set by solving the linear system in (3.4). This initial model is shown in Fig. 1(a).

After the first run network training (solving (4.2) using the Adam solver), the network adjusts its parameters to adapt the target function f. The resulting optimized network model with 10 neurons is shown in Fig. 1(b). This NN model provides a near-optimum free-knot piecewise linear spline with 10 knots shown as the green break points in the Figure. Base on the partition of the domain with current set of break points, we adopt the average marking strategy (5.5) to mark the elements with errors larger than average, and then add neurons accordingly by setting the newly added neuron's initial biases at the centers of the elements to be refined. We then resolve for the new output layer's parameters using (3.4) and trained the network for the second run. This process repeats until the approximation error is lower than the target ϵ. The ANE method iterates itself three runs from 10 to 13 then to 20 neurons. The intermediate result at 13 neurons is depicted in Fig. 1(c). The ANE pro-



(a) 10 neurons (with three marked elements)

(b) 13 neurons (with seven marked elements

(c) 20 neurons

Fig. 2. Error distribution on physical partitions generated in the ANE process for the first test problem, where red partitions are the elements to be refined.

Table 1
Comparing adaptive neural network with fixed networks for testing problem *(7.1)*.

Network (neurons)	# Parameters	$\|f - f_r\|_r / \|f\|$
Fixed (20)	41	0.007644
Fixed (38)	77	0.003762
Adaptive (10→13→20)	41	0.003837

cess ends at 20 neurons, which gives a relative approximation accuracy of $\xi = 0.003837$, falling below the target ϵ.

In this one-dimensional problem, we utilize a fixed learning rate of 0.001. Fig. 2(a) and 2(b) show the error per element distribution on the physical partition generated through the iterative process. In those two figures, red bars correspond with marked elements where new neurons are to be added. Marked red elements are refined and the iterative process gradually drags all elements error down to a smaller scale with a trend to distribute the error evenly among the physical partitions, see error distribution of the final network model in Fig. 2(c).

We further compare the performance of our adaptive network structure with a network model of fixed number of neurons. This is to check if the adaptive process has a potential to land in a better global minimum. The comparison results are illustrated in Table 1 and Fig. 3. In particular, they show that approximation accuracy of the adaptive network using ANE with 20 neurons is almost same as that of the fixed network with 38 neurons, and is better than the fixed network structure of the same size. This experiment indicates that fix networks might tend to be trapped in local minimums.

Finally, we test the performance of enhancement strategy using (5.3), and compare two methods of initialization under the global adaptive enhancement scheme. The first initialization method is to add new neurons randomly and set their corresponding output weights as zeros as initial; and the second method is to add new neurons uniformly across the domain and solve (3.4) for output weights and bias. Table 2 list the results of this experiment. Due to the non-convex optimization, one can see that different initialization strategy result in differences in approximation results. The first initialization method is easily trapped in local minimum under the Adam optimizer. Considering the randomness in the initial of newly added neurons, we repeat this test three times and report the best result in the table. While in the second strategy, we start from a better point using global uniform refinement, this results in a better performance. However, the uniform initial strategy during the adaptive process does not consider the error distribution evaluated from the previous stage, which explains why it is still inferior to the local error based marking and refinement strategy. For the rest experiments, we only use the local enhancement method.

7.2. Functions with intersecting interface singularities

This section reports the numerical results for a two-dimension problem with intersecting interface singularity. Let $\Omega = (-1, 1)^2$ and

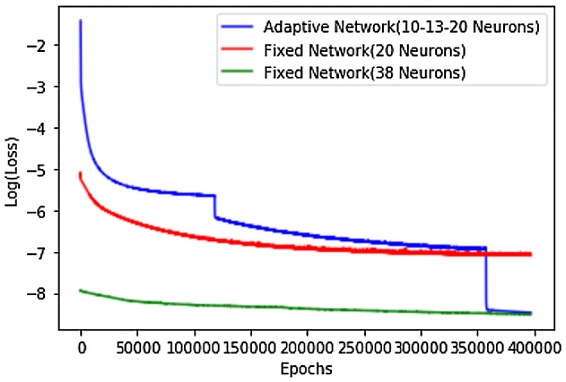


Fig. 3. Log training loss with three different network models in the first numerical experiment.

Table 2
Global network enhancement and initialization strategy for testing problem *(7.1)*.

Network (adaptive neurons)	Initialization	# Parameters	$\|f - f_r\|_r / \|f\|$
10→20→40	random	81	0.005221
10→20→30	uniform	61	0.004455

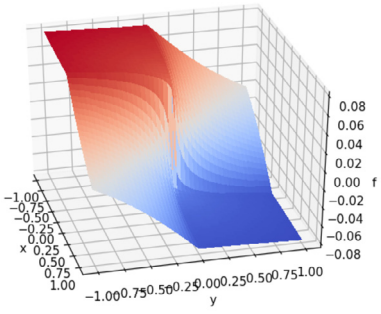
$$f(r, \theta) = r^{\beta}\mu(\theta) \tag{7.2}$$

in the polar coordinates at the origin with

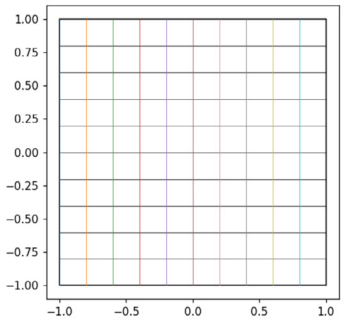
$$\mu(\theta) = \begin{cases} \cos((\pi/2 - \sigma)\beta) \cdot \cos((\theta - \pi/2 + \rho)\beta), & \text{if } 0 \le \theta \le \pi/2, \\ \cos(\rho\beta) \cdot \cos((\theta - \pi + \sigma)\beta), & \text{if } \pi/2 \le \theta \le \pi, \\ \cos(\theta\beta) \cdot \cos((\theta - \pi - \rho)\beta), & \text{if } \pi \le \theta \le 3\pi/2, \\ \cos((\pi/2 - \rho)\beta) \cdot \cos((\theta - 3\pi/2 - \sigma)\beta), & \text{if } 3\pi/2 \le \theta \le 2\pi, \end{cases}$$

where $\beta = 0.1$, $\sigma = -14.92256510455152$, and $\rho = \pi/4$ are parameters. The function $f(r, \theta)$ (see Fig. 4(a)) is the solution of the elliptic interface problem with intersecting interface singularity and a benchmark test problem for adaptive finite element method (see, e.g., [25,26]).

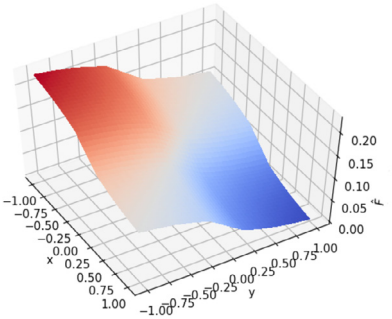
We test the ANE method with a fixed integration mesh using 400×400 quadrature points. The target approximation accuracy is set as $\epsilon = 0.01$. The ANE process starts with a small network of 20 neurons, and the network is initialized such that the break lines are distributed evenly in the domain, with half of them parallel to x-axis ($\boldsymbol{\omega}_i = 0$ and $b_i = -1 + 0.2i$ for $i = 0, \cdots, 9$) and the other half parallel to y-axis ($\boldsymbol{\omega}_i = \pi/2$ and $b_i = -1 + 0.2(i - 10)$ for $i = 10, \cdots, 19$). See Fig. 4(b) for the initial partition of the domain. The initial network model using this uniform physical partition is obtained by solving the linear system in (3.4) and is shown in Fig. 4(c). After the first run network training, the optimum break lines corresponding to the 20-neuron two-layer ReLU network are shown in Fig. 4(d) and the corresponding network model is plotted in Fig. 4(e). With 20 neurons (61 parameters), the adaptive network



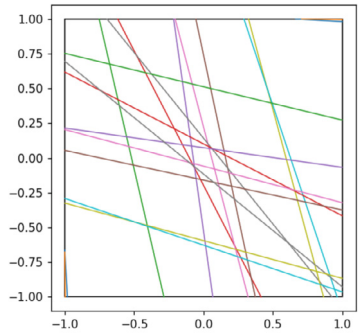
(a) Target function f(x,y)



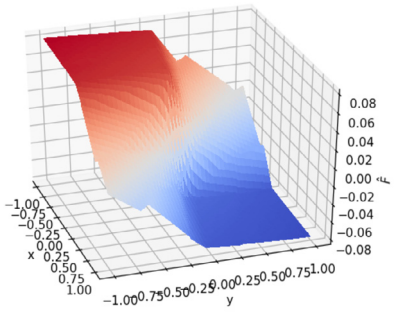
(b) Initial break lines (20 neurons, 100 elements)



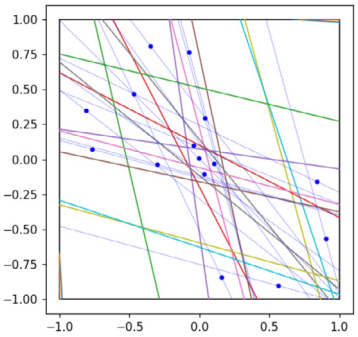
(c) Initial NN model with 20 neurons



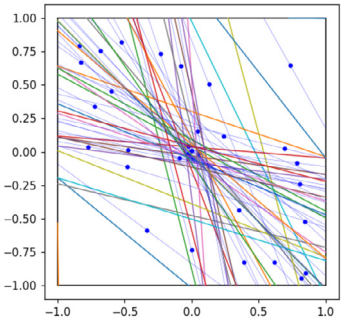
(d) Optimum break lines (20 neurons, 103 elements)



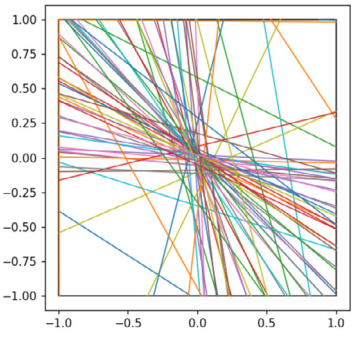
(e) Optimum NN model of 20 neurons, $\xi = 0.038733$



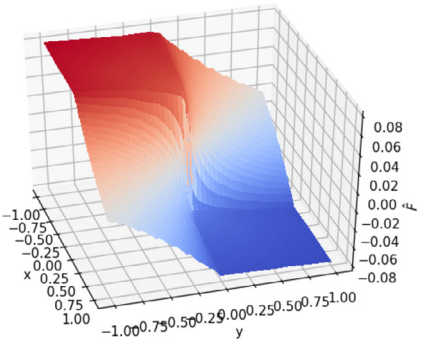
(f) 15 more neurons added with break lines in light blue



(g) Optimum break lines of 35 neurons and 34 more neurons are to be added in the second run



(h) Optimum break lines (69 neurons, 1286 elements)



(i) Optimum NN model of 69 neurons, $\xi = 0.008476$

Fig. 4. ANE results of using 2-layer ReLU networks for approximating function in (7.2).

can approximate the target function f in (7.2) with a relative error $\xi = 0.038733$.

To achieve the target accuracy, the ANE calculates per element error base on the automatic generated physical partition of the domain. Elements with relative large errors are marked using the bulk marking strategy (5.6) with $\gamma_1 = 0.7$ (see the 15 elements with blue dots shown in Fig. 4(f)). ANE process adds the same number of neurons as the 15 marked elements, and those new neurons are initialized as follows: their corresponding breaking lines pass through the centroids of marked elements, with their directions aligned with the maximum principal directions of each geometric element. See Fig. 4(f) for the initial physical partition at the second run with the newly added neuron's breaking lines drawn in light blue. The second run network training converged at a relative error $\xi = 0.019582$ (see the generated physical partition and marked elements in Fig. 4(g)). The ANE process stops at 69 neurons with the corresponding physical partition and network model plotted in Fig. 4(h) and Fig. 4(i). Notice that to calculate per element error, and to find an element's centroid and principal direction, we group the quadrature points located in the same element and use the point set within the element to compute its local error, centroid and PCA. This approximation method has an advantage of its computational simplicity; by avoiding calculation of the exact geometric shape of each element, this method can be easily extended to higher dimension problems or higher order activation functions.

Table 3
The effect of numerical integration for the second testing problem (7.2).

Network (# quadrature)	Integration accuracy $\lvert(\mathcal{I}-\mathcal{Q})(f)\rvert/\lvert\mathcal{I}(f)\rvert$	Training accuracy $\lVert f-f_r\rVert_r/\lVert f\rVert$	Testing accuracy $\lVert f-f_r\rVert_{r'}/\lVert f\rVert$
Fixed (50x50)	0.002638	0.007885	0.013187
Fixed (100x100)	0.000753	0.008515	0.010257
Fixed (200x200)	0.000462	0.009319	0.009877
Fixed (400x400)	0.000370	0.009702	0.009850
ANE (400x400)	0.000370	0.008319	0.008476

A fixed learning rate of 10^{-3} is adopted in this ANE process. The final network model achieves a L^2 relative error of $\xi = 0.008476$, which meets our approximation accuracy target. The generated physical partition is highly adapted to the target function. Notice there is a point singularity around the origin in the function f, while the physical partition obtained in the adaptive network adjusts its elements shape and size such that the partition is dense around the singular point, this is a very favorable property of using NN model to approximate functions with singularities. Comparing with adaptive finite element methods (AFEMs) (see, e.g., [26]), the ANE method has much fewer degrees of freedom than AFEMs.

To evaluate the effect of numerical integration to the total approximation error, we tested a two-layer network of 69 neurons using varying $\mathcal{T}$ with different number of quadrature points. The results are given in Table 3. As shown in the table, with finer integration meshes of more number of quadrature points, the integration accuracy can be improved (refer to the 'Integration accuracy' column in the table). Meanwhile, training a network model on finer mesh is harder which results in a lower training accuracy (see the 'Training accuracy' column). However, the approximating power to the true function f is improved (see the 'Testing accuracy' column in Table 3). Notice here the testing accuracy is estimated using a fine mesh $\mathcal{T}'$ of 1000×1000 quadrature points. The gap between training accuracy and testing accuracy is reduced when more number of quadrature points is adopted. This experiment also shows that the adaptive network may achieve better approximation result compared with the fixed network of the same size, see the last two rows in Table 3.

7.3. Functions with transition layers

The last problem we tested is a two-dimensional function with a transition layer around a circular region:

$$f(x,y)=\tanh\left(\frac{1}{\varepsilon}(x^2+y^2-\frac{1}{4})\right)-\tanh\left(\frac{3}{4\varepsilon}\right) \tag{7.3}$$

defined on the domain $\Omega=[-1,1]^2$. By varying ε, this type of functions shows different level of difficulties due to the presence of transition layers. We set $\varepsilon = 0.01$ in this experiment, and the corresponding function f presents a large transition in a sharp circular zone, as shown in Fig. 5(a).

For this problem, we ran three tests to compare the results of using an uniform integration mesh versus adaptive mesh refinement (AMR). (1) The first experiment utilizes an uniformly allocated $400\times400=1.6\times10^5$ quadrature points and the ANE Algorithm 5.1 to obtain a network model of 578 neurons with target accuracy $\epsilon = 0.05$. (2) The second experiment uses Algorithm 5.3 which generates an AMR of $22201\approx2.2\times10^4$ quadrature points (as shown in Fig. 5(b)) and an adaptive NN of 578 neurons as well. The 22201 quadrature points are generated by adaptive local mesh refinement of an initial mesh of 100×100 quadrature points, using average marking strategy. We set the last run ANE process to stop at 578 neurons to allow a fair comparison to the first experiment. (3) the third experiment matches the number of quadrature points used in the second experiment, but with those 150×150 quadrature points allocated uniformly across the domain, and a fixed network model of 578 neurons was tested to compare the approximation performances with the ANE network using AMR integration mesh.

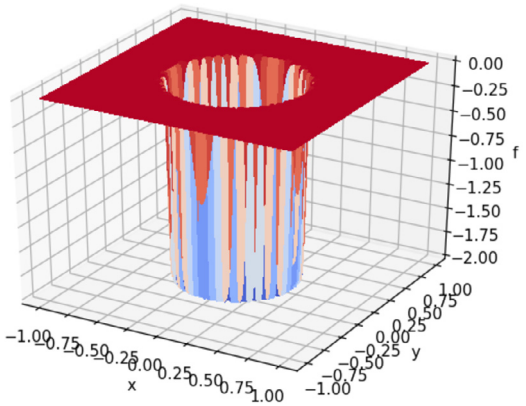
The comparison results are illustrated in Table 4. The ANE method using AMR for numerical integration achieves better performance compared with a finer uniform mesh of six times more quadrature points and it is superior compared with the similar mesh size but evenly distributed quadrature points. If limited computational resources are allocated which allows only certain number of quadrature points for numerical integration and network training, allocating quadrature points using AMR might achieve better approximation performance compared with the uniformly allocated quadrature points.

The function approximation result shown in Fig. 5(c) exhibits a certain level of oscillation which is not acceptable in some applications. Notice that the generated physical partition (see Fig. 5(d)) does capture the circular transition layers well when using 578 break lines. However, this partition is too dense in the region where the function does not fluctuate much. A deeper ReLU network, which provides piece-wise breaking lines, might work better for this testing case. We verified this conjecture by using a three-layer ReLU network to approximate this function. Each hidden layer was set as fixed 20 neurons which defines a network model of 501 parameters. The relative approximation error ξ using this three-layer ReLU network is 0.033967. Comparing to the 578 neurons and 1735 parameters we used previously in the two-layer networks, a three-layer ReLU network of smaller size can approximate the same function with better accuracy (see Table 5). As illustrated in Fig. 6(a), a three-layer network can reduce the oscillation exhibited in the shallow network, while archiving a better approximation accuracy with less complicated domain partition, see Fig. 6(b) for the physical partition generated with the three-layer network. This experiment gives us insights for our follow-up work [27] on an adaptive network enhancement method which will study the problem of generating multi-layer networks, in terms of both width and depth, in order to approximate functions/PDEs of different characteristics accurately and efficiently.

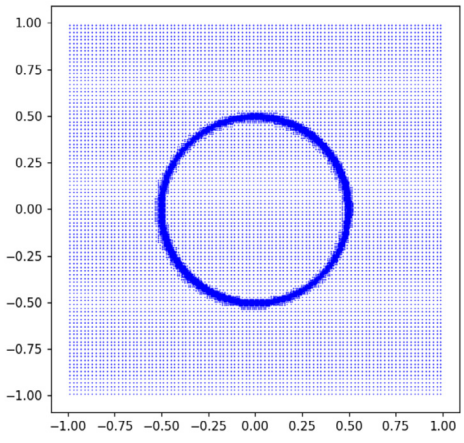
8. Discussion and conclusion

This paper studies a fundamental question in machine learning on how to design the architecture of two-layer neural networks in order to approximate functions accurately and efficiently. For a given function, we introduce and test an adaptive network enhancement (ANE) method that adaptively constructs a two-layer NN with a relatively small number of neurons and parameters such that its approximation accuracy is within the prescribed tolerance. One of key components of the ANE method for the best least-squares approximation to a given function is the enhancement strategy which determines how many new neurons to be added, when the current approximation is not within the given accuracy. To address this issue, a global and a local network enhancement strategies are introduced and tested. The efficacy of the local enhancement strategy is demonstrated numerically for several test problems in this paper. Due to uncertainty of non-convex optimization, numerical results also show that the local strategy is better than the global one. Nevertheless, efficiency and robustness of both the global and local enhancement strategies need further numerical and theoretical studies.

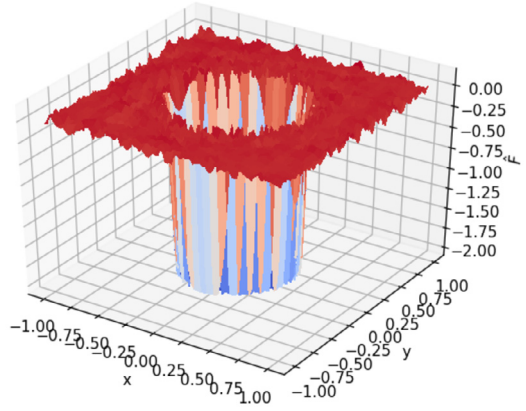
To disentangle the numerical integration error and network approximation error, an AMR method is proposed for automatically generating an integration mesh which adapts itself to improve the numerical integration accuracy. The AMR method presented in the paper is suitable for



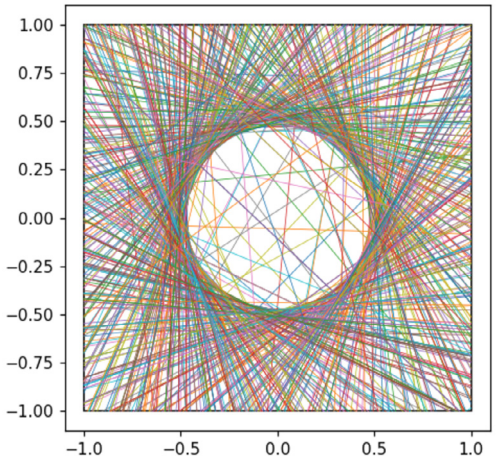
(a) The target function f with a circular transition layer



(b) The generated AMR of 22201 quadrature points



(c) ANE network model of 578 neurons using AMR in (b) for training (integration), $\xi = 0.048771$



(d) The physical partition generated by ANE of 578 neurons

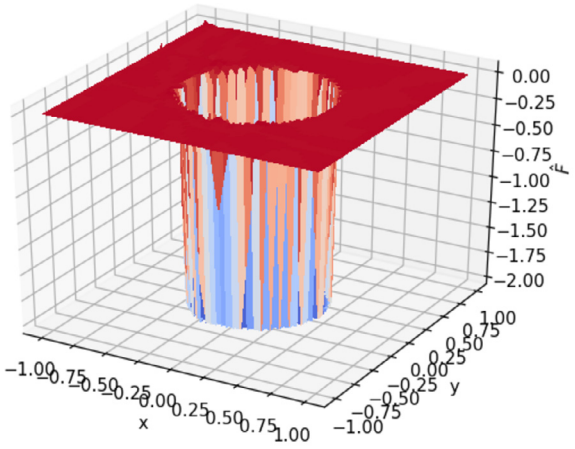
Fig. 5. ANE with AMR results of using 2-layer ReLU networks for approximating function in (7.3).

Table 4
Networks approximation performances of uniform v.s. AMR integration mesh.

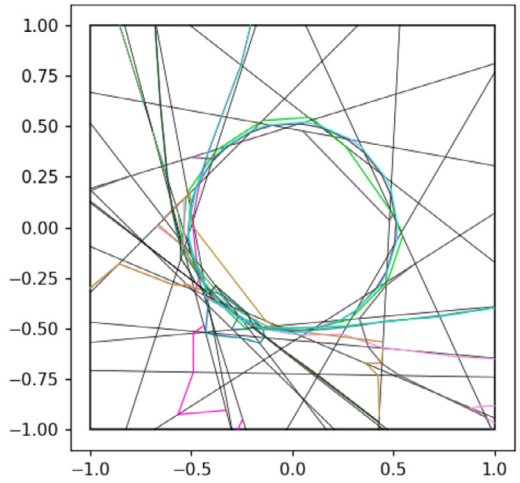
Integration mesh	# quadrature	# neurons	Training accuracy $\|f - f_r\|_r / \|f\|$	Testing accuracy $\|f - f_{r'}\|_{r'} / \|f\|$
Uniform	400x400	ANE 578	0.050552	0.050587
AMR	22201	ANE 578	0.047423	0.048771
Uniform	150x150	Fixed 578	0.052497	0.053040

Table 5
Approximation performances of a two-layer v.s. a three-layer NN.

NN structure (neurons)	#Quadrature	#Parameters	Training accuracy $\|f - f_r\|_r / \|f\|$	Testing accuracy $\|f - f_{r'}\|_{r'} / \|f\|$
Two-layer (578)	AMR 22201	1735	0.047423	0.048771
Three-layer (20-20)	uniform 150x150	501	0.033751	0.033969



(a) a three-layer network model of 20 neurons in each hidden layer, $\xi = 0.033967$



(b) Physical partition generated by the three-layer network (black lines are the break lines in the first hidden layer, colored lines are the break polylines in the second hidden layer

Fig. 6. Approximation results of using a three-layer ReLU network for approximating function in (7.3).

low dimensional problems and may be replaced by any adaptive integration procedure such as adaptive version of Monte Carlo, quasi-Monte Carlo, or sparse grid, etc. if a high dimensional problem is considered. Nevertheless, for a given function, how to adaptively choose a proper numerical integration in the context of NN functions remains open and requires further investigation.

Determining the values of the parameters of NNs is a problem in non-convex optimization which is computationally intensive and complicated and is a bottleneck in using NNs. Commonly used iterative solvers for optimization in NN applications are iterative methods of the gradient descent type. It is a common sense that it is extremely difficult, if not impossible, to develop a computationally feasible iterative solver that would converge to the desired global optimizer. This, in turn, implies the prominent importance of a close enough first approximation for all iterative solvers, as experienced in our numerical experiments. The method of continuation [28] is a common way to obtain a good initial and the ANE is a natural continuation process by itself with respect to the number of neurons. In particular, weights and bias of newly added neurons are initialized based on the implicit physical partition of the domain Ω for the NN approximation at the previous network. This deterministic initialization strategy ensures that the starting point of each iteration is always superior to the previous iteration when the network is enhanced, and plays an essential role in training the current network.

Experimental results for functions exhibiting intersecting interface singularities or sharp interior layer like discontinuities show the efficacy of the propose method. In the second part of the paper [7], we extend the application of the proposed ANE method to elliptic partial differential equation with an underlying minimization principle.

References

[1] D.O. Hebb, The Organization of Behavior: A Neuropsychological Theory, Wiley, New York, 1949.
[2] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386–408.
[3] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[4] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[5] P.P. Petrushev, Approximation by ridge functions and neural networks, SIAM J. Math. Anal. 30 (1998) 155–189.
[6] A. Pinkus, Approximation theory of the mlp model in neural networks, Acta Numer. 8 (1999) 143–195.
[7] M. Liu, Z. Cai, Adaptive two-layer ReLU neural network II: Ritz approximation to elliptic PDEs, arXiv:2107.06459 [math.NA], 2021.
[8] J. Dick, F. Kuo, I. Sloan, High-dimensional integration - the quasi-Monte Carlo way, Acta Numer. 15 (2014) 133–288.
[9] H.J. Bungartz, M. Griebel, Sparse grids, Acta Numer. 13 (2004) 1–123.
[10] L. Bottou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2018) 223–311.
[11] L. Schumaker, Spline Functions: Basic Theory, 1981.
[12] D. Jupp, Approximation to data by splines with free knots, SIAM J. Numer. Anal. 15 (6) (1978) 328–343.
[13] A.J. Baker, On optimization aspects of a cfd finite element penalty algorithm, in: J.R. Whiteman (Ed.), The Mathematics of Finite Element and Applications V, 1985, pp. 391–414.
[14] J.R. Rice, The Approximation of Functions, vol. 2, Addison-Wesley, MA, 1969.
[15] M. Powell, On best l 2 spline approximations, Num. Math. Differ. Approx. (1968) 317–339.
[16] C.K. Chui, P.W. Smith, J.D. Ward, On the smoothness of best L2 approximants from nonlinear spline manifolds, Math. Comput. 31 (1977) 17–23.
[17] M. Dissanayake, N. Phan-Thien, Neural network based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[18] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1139–1364.
[19] 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.
[20] 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.
[21] V.N. Temlyakov, The Marcinkiewicz-type discretization theorems, Constr. Approx. 48 (2018) 337–369.
[22] K. Pearson, On lines and planes of closest fit to systems of points in space, The London, Edinburgh, and Dublin Philos. Mag. J. Sci. 2 (11) (1901) 559–572.
[23] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[24] J. He, L. Li, J. Xu, C. Zheng, Relu deep neural networks and linear finite elements, J. Comput. Math. 38 (3) (2020) 502–527.
[25] P. Morin, R.H. Nochetto, K.G. Siebert, Convergence of adaptive finite element methods, SIAM Rev. 44 (4) (2002) 631–658.
[26] Z. Cai, S. Zhang, Recovery-based error estimator for interface problems: conforming linear elements, SIAM J. Numer. Anal. 47 (3) (2009) 2132–2156.
[27] Z. Cai, J. Chen, M. Liu, Self-adaptive deep neural network: numerical approximation to functions and PDEs, arXiv:2109.02839 [math.NA], 2021.
[28] E. Allgower, K. Georg, Numerical Continuation Methods, Springer-Verlag, Berlin and Heidelberg, 1990.