SELF-ADAPTIVE RELU NEURAL NETWORK METHOD IN LEAST-SQUARES DATA FITTING*

ZHIQIANG CAI† AND MIN LIU‡

Abstract. This chapter provides a comprehensive introduction to a self-adaptive ReLU neural network method proposed recently in [11, 10, 5]. The purpose of the method is to design a nearly minimal neural network architecture to achieve the prescribed accuracy for a given task in scientific machine learning such as approximating a function or a solution of partial differential equation. Starting with a small one hidden-layer neural network, the method enhances the network adaptively by adding neurons in the current or new hidden-layer based on accuracy of the current approximation. In addition, the method provides a natural process for obtaining a good initialization in training the current network. Moreover, initialization of newly added neurons at each adaptive step is discussed in detail.

Key words. Self-adaptivity, Least-squares data fitting, Deep neural network, ReLU activation

1. Introduction. Given a data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^M$ with $\mathbf{x}_i \in \Omega = [-1,1]^d$ and positive weights $\{w_i\}_{i=1}^M$, consider the discrete least-squares problem: finding $f_{nn}(\mathbf{x}) \in \mathcal{M}(l)$ such that

$$f_{nn} = \arg\min_{v \in \mathcal{M}(l)} L(v), \tag{1.1}$$

where $\mathcal{M}(l)$ is a ReLU neuron network defined in section 2 with l hidden-layers and $L(\cdot)$ is a least-squares loss functional given by

$$L(v) = \sum_{i=1}^{M} w_i \left(v(\mathbf{x}_i) - y_i\right)^2.$$

For a prescribed tolerance $\epsilon > 0$, this chapter presents a self-adaptive algorithm, the adaptive neuron enhancement method (ANE), to adaptively construct a nearly optimal network $\mathcal{M}^*$ such that the neural network approximation $f_{nn}(\mathbf{x})$ satisfies

$$L(f_{nn}) \le \epsilon\, L(0), \tag{1.2}$$

where $L(0) = \sum_{i=1}^{M} w_i y_i^2$ is the square of the weighted l^2 norm of the output data $\{y_i\}_{i=0}^M$.

Multi-layer ReLU neural network is described in this chapter as a set of continuous *piece-wise* linear functions. Hence each network function is piece-wise linear with respect to a partition of the domain. This partition, referred as the (domain) physical partition (see section 3), provides geometric feature of the function and hence plays a critical role in the design of self-adaptive neural network method. Determination of this physical partition for a network function is in general computationally expensive, especially when the input dimension d is high. To circumvent this difficulty, we introduce a network indicator function that can easily determine such partition.

The idea of the ANE is similar to that of standard adaptive mesh-based numerical methods, and may be written as loops of the form

$$\textbf{train} \;\rightarrow\; \textbf{estimate} \;\rightarrow\; \textbf{mark} \;\rightarrow\; \textbf{enhance}. \tag{1.3}$$

*This work was supported in part by the National Science Foundation under grant DMS-2110571.

†Department of Mathematics, Purdue University, 150 N. University Street, West Lafayette, IN 47907-2067 (caiz@purdue.edu.

‡School of Mechanical Engineering, Purdue University, 585 Purdue Mall, West Lafayette, IN 47907-2088(liu66@purdue.edu).

Starting with a small one hidden layer network, the step **train** is to iteratively solve the optimization problem of the current network; the step **estimate** is to compute error of the current approximation; the step **mark** is to identify local regions that need refinement; and the step **enhance** is to add new neurons to the current network with good initialization. This adaptive algorithm learns not only from given information (data, function, partial differential equation) but also from the current computer simulation.

When the current error does not satisfy (1.2), an efficient ANE method relies on strategies to address the following questions at each adaptive step:

(a) how many new neurons should be added at the last hidden layer?

(b) when should a new hidden layer be added?

By exploiting the geometric feature of the current approximation, the enhancement strategy (see section 4) determines the number of new neurons to be added at the last hidden layer. A new layer is added if a computable quantity measuring the improvement rate of two consecutive networks per the relative increase of parameters is small.

Problem (1.1) is a non-convex optimization that has many solutions, and the desired one is only attainable when one begins with an initial approximation that is sufficiently close. A common approach to obtaining a good initialization is through the method of continuation, as described in [1]. The ANE method offers a natural way to acquire a well-suited initialization. Essentially, the approximation provided by the previous network serves as a good starting point for the current network at each adaptive step. Additionally, we outline an approach for initializing the weights and biases of newly added neurons, leveraging the geometric properties of the current approximation, which is detailed in section 5.

2. ReLU Neural Network. A neural network defines a new class of approximating functions which is suitable for some computationally challenging problems. This section describes l-hidden-layer ReLU neural network as a set of continuous piece-wise linear functions and introduces related notations. This chapter is restricted to one dimensional output $n_{l+1} = 1$ for simplicity of presentation. Extension of materials covered by this chapter to multi-dimensional output $n_{l+1} > 1$ is straightforward.

ReLU refers to the rectified linear activation function defined by

$$\sigma(t) = \max\{0, t\} = \begin{cases} t, & t > 0, \\ 0, & t \le 0. \end{cases} \tag{2.1}$$

The $\sigma(t)$ is a continuous piece-wise linear function with one *breaking* point $t = 0$. For $k = 1, \ldots, l$, let n_k denote the number of neurons at the k^{th} hidden-layer; denote by

$$\mathbf{b}^{(k)} \in \mathbb{R}^{n_k} \quad \text{and} \quad \boldsymbol{\omega}^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$$

the biases and weights of neurons at the k^{th} hidden-layer, respectively. Their i^{th} rows are denoted by $b_i^{(k)} \in \mathbb{R}$ and $\boldsymbol{\omega}_i^{(k)} \in \mathbb{R}^{n_{k-1}}$, that are the bias and weights of the i^{th} neuron at the k^{th} hidden-layer, respectively. Introduce a vector-valued function $\mathbf{N}^{(k)} : \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$ as

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

where application of the activation function σ to a vector-valued function is defined component-wisely and $n_0 = d$ is the input dimension.

A ReLU neural network with l hidden-layers and n_k neurons at the k^{th} hidden-layer may be defined as the collection of continuous piece-wise linear functions:

$$\mathcal{M}(l) = \left\{ \mathbf{c}_1\left(\mathbf{N}^{(l)} \circ \cdots \circ \mathbf{N}^{(1)}(\mathbf{x})\right) + c_0 : \begin{array}{l} (c_0, \mathbf{c}_1) \in \mathbb{R}^{n_l+1},\ \boldsymbol{\omega}^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}, \\ \mathbf{b}^{(k)} \in \mathbb{R}^{n_k} \ \text{ for } \ k = 1, \ldots, l \end{array} \right\}, \tag{2.3}$$

where the symbol $\circ$ denotes the composition of functions. The total number of parameters of $\mathcal{M}(l)$ is given by

$$M(l) = (n_l + 1) + \sum_{k=1}^{l} n_k \times (n_{k-1} + 1). \tag{2.4}$$

As in [4], the biases and weights of all hidden-layers

$$\boldsymbol{\Theta}^{(l)} = \bigcup_{k=1}^{l} \left\{ \left(b_i^{(k)}, \boldsymbol{\omega}_i^{(k)} \right) \right\}_{i=1}^{n_k} = \left\{ \left(\mathbf{b}^{(k)}, \boldsymbol{\omega}^{(k)} \right) \right\}_{k=1}^{l} \tag{2.5}$$

are referred as nonlinear parameters, and the output bias and weights

$$\mathbf{c} = (c_0, \mathbf{c}_1) = (c_0, c_1, \ldots, c_{n_l}) \in \mathbb{R}^{n_l+1}$$

are referred as linear parameters for a neural network function.

REMARK 2.1. *Domain of the nonlinear parameter $\boldsymbol{\Theta}^{(l)}$ in (2.3) is too large in general and hence admit infinite many global minimizers of (1.1). One may add some constraints to the domain in order to reduce the number/dimension of the global minimizers. For example, the weights of each neuron can be normalized (see, e.g., [4, 11, 8]).*

Linearity of the output parameter $\mathbf{c}$ here means that $\mathbf{c}$ is uniquely determined by a system of linear algebraic equations with given nonlinear parameter $\boldsymbol{\Theta}^{(l)}$. In the remainder of this section, we introduce this linear system and show that the corresponding mass matrix is always symmetric; moreover, it is positive definite under some condition. To this end, let

$$\varphi_0^{(l)}(\mathbf{x}) = 1 \quad \text{and} \quad \varphi_i^{(l)}(\mathbf{x}) = \sigma\left(\boldsymbol{\omega}_i^{(l)} \left(\mathbf{N}^{(l-1)} \circ \cdots \circ \mathbf{N}^{(1)}(\mathbf{x}) \right) + b_i^{(l)} \right), \tag{2.6}$$

then any function $v \in \mathcal{M}(l)$ has the form of

$$v(\mathbf{x}) = \sum_{i=0}^{n_l} c_i \varphi_i^{(l)}(\mathbf{x}). \tag{2.7}$$

A solution f_{nn} of (1.1) satisfies the critical point equation

$$\nabla_{\mathbf{c}} L(f_{nn}) = \mathbf{0} \tag{2.8}$$

for the linear parameter $\mathbf{c} = (c_0, \mathbf{c}_1)$. This implies that $\mathbf{c}$ satisfies the following system of linear algebraic equations

$$\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right) \mathbf{c} = F^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right), \tag{2.9}$$

where $\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$ and $F^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$ are the discrete mass matrix and the right-hand side vector given by

$$\begin{cases} \boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right) = \left(\sum_{e=1}^{M} w_e \varphi_j^{(l)}(\mathbf{x}_e) \varphi_i^{(l)}(\mathbf{x}_e) \right)_{(n_l+1)\times(n_l+1)} & \text{and} \\ F^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right) = \left(\sum_{e=1}^{M} w_e y_e \varphi_i^{(l)}(\mathbf{x}_e) \right)_{(n_l+1)\times 1}. \end{cases} \tag{2.10}$$

LEMMA 2.2. *The mass matrix* $\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$ *defined in (2.10) is symmetric. Assume that functions* $\left\{\varphi_i^{(l)}(\mathbf{x})\right\}_{i=0}^{n_l}$ *are linearly independent, then* $\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$ *is positive definite.*

Proof. Obviously, $\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$ is symmetric. For any $\mathbf{c}\in\mathbb{R}^{n_l+1}$, we have

$$\mathbf{c}^T\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)\mathbf{c}=\sum_{i,j=0}^{n_l+1}\sum_{e=1}^{M}c_ic_jw_e\varphi_j^{(l)}(\mathbf{x}_e)\varphi_i^{(l)}(\mathbf{x}_e)=\sum_{e=1}^{M}w_e\left(\sum_{i=0}^{n_l+1}c_i\varphi_i^{(l)}(\mathbf{x}_e)\right)^2,$$

which, together with the assumption, implies positive definiteness of $\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$. ∎

Even though $\boldsymbol{M}^{(l)}\left(\boldsymbol{\Theta}^{(l)}\right)$ is symmetric, positive definite, it could be highly ill-conditioned. This fact, in turn, implies inefficiency of the optimization methods of gradient descent type.

3. Physical Partition. A neural network function in $\mathcal{M}(l)$ has the form of

$$v(\mathbf{x})=\mathbf{c}_1\left(\mathbf{N}^{(l)}\circ\cdots\circ\mathbf{N}^{(1)}(\mathbf{x})\right)+c_0=\sum_{i=0}^{n_l}c_i\varphi_i^{(l)}(\mathbf{x}),\tag{3.1}$$

where $\varphi_i^{(l)}(\mathbf{x})$ is defined in (2.6). Obviously, $v(\mathbf{x})$ is a continuous *piece-wise* linear (CPWL) function defined in $\mathbb{R}^d$. This means that there exists a partition of $\mathbb{R}^d$ such that $v(\mathbf{x})$ is linear on all subdomains of this partition. This section studies such a partition for a given neural network function $v(\mathbf{x})$ of the form in (3.1).

DEFINITION 3.1. *For a given network function* $v(\mathbf{x})$ *of the form in (3.1) defined in* $\Omega=[-1,1]^d$*, a partition* $\mathcal{K}^{(l)}(v)$ *of* Ω *is said to be the physical partition of* $v(\mathbf{x})$ *with respect to* Ω *if*

(i) $\mathcal{K}^{(l)}(v)$ *is a partition of* Ω*, i.e.,*

$$\Omega=\bigcup_{K\in\mathcal{K}^{(l)}(v)}\bar{K}\quad\text{and}\quad K\cap T=\emptyset\ \text{if}\ K\neq T\ \text{for all}\ K,T\in\mathcal{K}^{(l)}(v).$$

(ii) *for each subdomain* $K\in\mathcal{K}^{(l)}(v)$*, the restriction of* $v(\mathbf{x})$ *on* K *is a linear function.*

REMARK 3.2. *The physical partition* $\mathcal{K}^{(l)}(v)$ *defined in Definition 3.1 depends on the nonlinear parameter* $\boldsymbol{\Theta}^{(l)}$ *but not on the linear parameter* $\mathbf{c}$*.*

For a shallow neural network $\mathcal{M}(1)$, each function $v\in\mathcal{M}(1)$ has the form of

$$v(\mathbf{x})=\mathbf{c}_1\left(\mathbf{N}^{(1)}(\mathbf{x})\right)+c_0=\sum_{i=1}^{n_1}c_i\sigma\left(\boldsymbol{\omega}_i^{(1)}\mathbf{x}+b_i^{(1)}\right)+c_0=\sum_{i=0}^{n_1}c_i\varphi_i^{(1)}(\mathbf{x}).$$

where $\boldsymbol{\Theta}^{(1)}=\left\{\boldsymbol{\theta}_i^{(1)}\right\}_{i=1}^{n_1}:=\left\{\left(b_i^{(1)},\boldsymbol{\omega}_i^{(1)}\right)\right\}_{i=1}^{n_1}$ is nonlinear parameter. For $i=1,\ldots,n_1$, denote the pre-activation function of the i^{th} neuron by

$$g_i^{(1)}(\mathbf{x})=\boldsymbol{\omega}_i^{(1)}\mathbf{x}+b_i^{(1)}\tag{3.2}$$

and its zero level set, called the *breaking hyper-plane*, by

$$\mathcal{P}_i^{(1)}\left(\boldsymbol{\theta}_i^{(1)}\right)=\left\{\mathbf{x}\in\Omega:\,g_i^{(1)}(\mathbf{x})=0\right\}=\left\{\mathbf{x}\in\Omega:\,\boldsymbol{\omega}_i^{(1)}\mathbf{x}+b_i^{(1)}=0\right\}.\tag{3.3}$$

For fixed $\boldsymbol{\Theta}^{(1)}$, the physical partition $\mathcal{K}^{(1)}(v)$ is formed by the set of the breaking hyper-planes $\left\{\mathcal{P}_i^{(1)}\right\}_{i=1}^{n_1}$ and the boundary of the domain Ω.

The breaking hyper-planes $\left\{\mathcal{P}_i^{(1)}\right\}_{i=1}^{n_1}$ in one dimension ($d=1$) degenerate to the breaking points $\left\{-\dfrac{b_i^{(1)}}{\omega_i^{(1)}}\right\}_{i=1}^{n_1}$ (blue dots in Fig. 1(a)), which partitions the interval $\Omega=[-1,1]$ into sub-intervals. The breaking hyper-planes in two dimensions ($d=2$) degenerate to the breaking lines (blue lines in Fig. 1(b))

$$\mathcal{P}_i^{(1)}\left(\boldsymbol{\theta}_i^{(1)}\right)=\left\{\mathbf{x}=(x_1,x_2)\in\Omega=[-1,1]^2:\,\omega_{i1}^{(1)}x_1+\omega_{i2}^{(1)}x_2+b_i^{(1)}=0\right\},$$

which partition the domain $\Omega\in\mathbb{R}^2$ into irregular, polygonal sub-domains (see also Fig. 2(c)).

For a two-hidden-layer neural network $\mathcal{M}(2)$, each function $v\in\mathcal{M}(2)$ has the form of

$$v(\mathbf{x})=\mathbf{c}_1\left(\mathbf{N}^{(2)}\circ\mathbf{N}^{(1)}(\mathbf{x})\right)+c_0=\sum_{i=0}^{n_2}c_i\varphi_i^{(2)}(\mathbf{x}),$$

where $\left\{\varphi_i^{(2)}(\mathbf{x})\right\}_{i=1}^{n_2}$ are similarly defined as in (2.6) by

$$\varphi_0^{(2)}(\mathbf{x})=1\quad\text{and}\quad\varphi_i^{(2)}(\mathbf{x})=\sigma\left(\boldsymbol{\omega}_i^{(2)}\left(\mathbf{N}^{(1)}(\mathbf{x})\right)+b_i^{(2)}\right)=\sigma\left(\boldsymbol{\omega}_i^{(2)}\sigma\left(\boldsymbol{\omega}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+b_i^{(2)}\right)$$

for $i=1,\ldots,n_2$ and nonlinear parameters are given by

$$\boldsymbol{\Theta}^{(2)}=\boldsymbol{\Theta}^{(1)}\cup\left\{\boldsymbol{\theta}_i^{(2)}\right\}_{i=1}^{n_2}:=\boldsymbol{\Theta}^{(1)}\cup\left\{\left(b_i^{(2)},\boldsymbol{\omega}_i^{(2)}\right)\right\}_{i=1}^{n_2}.\tag{3.4}$$

Similar to the shallow network $\mathcal{M}(1)$, denote pre-activation functions of neurons at the 2^{nd} hidden-layer by

$$g_i^{(2)}(\mathbf{x})=\boldsymbol{\omega}_i^{(2)}\sigma\left(\boldsymbol{\omega}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+b_i^{(2)}\quad\text{for } i=1,\ldots,n_2\tag{3.5}$$

and their zero level sets, called the *breaking poly-hyper-planes*, by

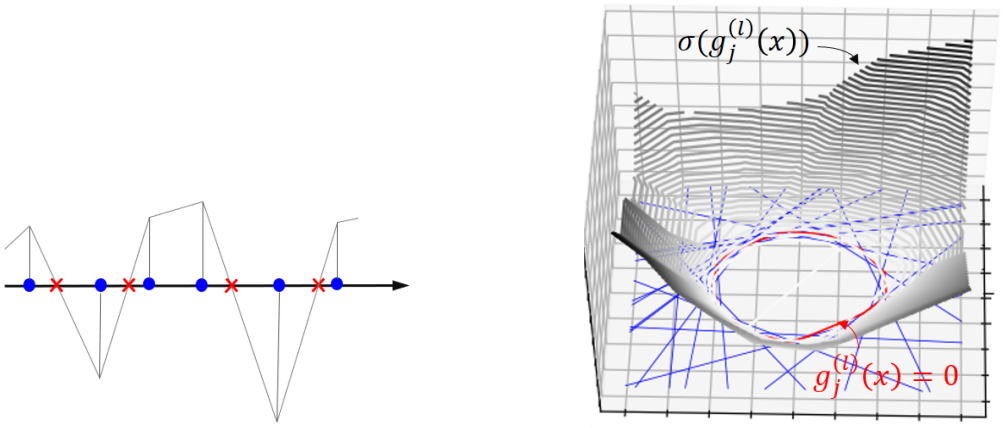
$$\mathcal{P}_i^{(2)}\left(\boldsymbol{\Theta}^{(1)},\boldsymbol{\theta}_i^{(2)}\right)=\left\{\mathbf{x}\in\Omega:\,\boldsymbol{\omega}_i^{(2)}\sigma\left(\boldsymbol{\omega}^{(1)}\mathbf{x}+\mathbf{b}^{(1)}\right)+b_i^{(2)}=0\right\}.\tag{3.6}$$

REMARK 3.3. *Note that $g_i^{(2)}(\mathbf{x})$ is a single-valued, continuous piece-wise linear function. This fact implies that $\mathcal{P}_i^{(2)}\left(\boldsymbol{\Theta}^{(1)},\boldsymbol{\theta}_i^{(2)}\right)$ as a zero level set is either empty or consists of poly-hyper-planes that do not intersect. Here, the poly-hyper-plane means a continuous hyper-plane that is composed of one or more connected hyper-plane segments. Moreover, each poly-hyper-plane is either closed or from part of the boundary to another part of the boundary.*

REMARK 3.4. *The physical partition $\mathcal{K}^{(2)}(v)$ is the refinement of the partition $\mathcal{K}^{(1)}(v)$ by using the breaking poly-hyper-planes $\left\{\mathcal{P}_i^{(2)}\left(\boldsymbol{\Theta}^{(1)},\boldsymbol{\theta}_i^{(2)}\right)\right\}_{i=1}^{n_2}$.*

In one dimension, $\mathcal{K}^{(2)}(v)$ is the refinement of $\mathcal{K}^{(1)}(v)$ by adding the 2^{nd} layer breaking points (red crosses in Fig. 1(a)) satisfying

$$\sum_{j=1}^{n_2}\omega_{ij}^{(2)}\sigma\left(\omega_j^{(1)}x+b_j^{(1)}\right)+b_i^{(2)}=0\quad\text{for } i=1,\ldots,n_2.$$



(a) Breaking points generated by the j^{th} neuron of the second layer

(b) Breaking lines generated by the j^{th} neuron of the l^{th}-layer

FIG. 1. *Breaking points/lines in the first two hidden layers.*

In two dimensions, the $\mathcal{K}^{(2)}(v)$ is the refinement of $\mathcal{K}^{(1)}(v)$ by adding the 2^{nd} layer breaking poly-lines (red poly-lines in Figs. 1(b) and 2(d)) satisfying

$$\sum_{j=1}^{n_2} \omega_{ij}^{(2)} \sigma\left(\boldsymbol{\omega}_j^{(1)}\mathbf{x} + b_j^{(1)}\right) + b_i^{(2)} = 0 \quad \text{for } i = 1, \ldots, n_2.$$

For $k = 1, \ldots, l-1$, denote the nonlinear parameter of the first k hidden-layers of $v(\mathbf{x})$ by

$$\boldsymbol{\Theta}^{(k)} = \boldsymbol{\Theta}^{(k-1)} \cup \left\{\boldsymbol{\theta}_i^{(k)}\right\}_{i=1}^{n_k} := \boldsymbol{\Theta}^{(k-1)} \cup \left\{\left(b_i^{(k)}, \boldsymbol{\omega}_i^{(k)}\right)\right\}_{i=1}^{n_k}. \tag{3.7}$$

Let $\mathcal{K}^{(k)}(v)$ denote the physical partition determined by the nonlinear parameters $\boldsymbol{\Theta}^{(k)}$. Then the physical partition $\mathcal{K}^{(l)}(v)$ may be described through a refinement process starting from the physical partition $\mathcal{K}^{(1)}(v)$. For $k = 2, \ldots, l$, the physical partition $\mathcal{K}^{(k)}(v)$ is the refinement of the previous physical partition $\mathcal{K}^{(k-1)}(v)$ by adding the following poly-hyper-planes

$$\mathcal{P}_i^{(k)}\left(\boldsymbol{\Theta}^{(k-1)}, \boldsymbol{\theta}_i^{(k)}\right) = \left\{\mathbf{x} \subset \mathbb{R}^d : g_i^{(k)}(\mathbf{x}) = 0\right\} \text{ for } i = 1, \ldots, n_k, \tag{3.8}$$

where $g_i^{(k)}(\mathbf{x})$ is the pre-activation function of the i^{th} neuron at the k^{th} hidden-layer given by

$$g_i^{(k)}(\mathbf{x}) = \boldsymbol{\omega}_i^{(k)}\left(\mathbf{N}^{(k-1)} \circ \cdots \circ \mathbf{N}^{(1)}(\mathbf{x})\right) + b_i^{(k)}. \tag{3.9}$$

The procedure for determining the physical partition $\mathcal{K}^{(l)}(v)$ of the domain Ω involves calculating the arrangement of a domain formed by a set of hyper-planes and poly-hyper-planes. This may be computationally expensive, especially when the input dimension d is high.

In practice, computation is usually done over a set of points in Ω, e.g., the input data set $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^M$ for problem (1.1) and integration point set as in [6, 10]. This motivates introduction of the *data physical partition*, i.e., the physical partition of $v(\mathbf{x}) \in \mathcal{M}(l)$ with respect to a given data set $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^M$. In a similar fashion as Definition 3.1, we define the data physical partition $\mathcal{D}^{(k)}$ as follows

$$\mathcal{D}^{(k)} = \mathcal{D} \cap \mathcal{K}^{(k)} = \left\{\mathcal{D} \cap K : K \in \mathcal{K}^{(k)}\right\} \tag{3.10}$$

for $k = 1, \ldots, l$.

Next, we describe how to form $\mathcal{D}^{(l)}$ for a given nonlinear parameters $\mathbf{\Theta}^{(l)}$. To this end, let $H(t)$ be the Heaviside step function given by

$$H(t) = \begin{cases} 1, & t > 0, \\ 0, & t < 0. \end{cases}$$

For $k = 1, \ldots, l$, introduce vector-valued *layer indicator function* $\mathbf{I}^{(k)} : \mathbb{R}^d \to \mathbb{R}^{n_k}$ as

$$\mathbf{I}^{(k)}(\mathbf{x}) = H\left(\mathbf{g}^{(k)}(\mathbf{x})\right), \tag{3.11}$$

where $\mathbf{g}^{(k)}(\mathbf{x}) = \left(g_i^{(k)}(\mathbf{x})\right)_{n_k \times 1}$ is the pre-activation function defined in (3.9) and application of H to a vector-valued function is defined component-wisely. For a given nonlinear parameter $\mathbf{\Theta}^{(l)}$, we define the *network indicator function* by

$$\mathcal{I}^{(l)}(\mathbf{x}) = \left(\mathbf{I}^{(1)}(\mathbf{x}), \ldots, \mathbf{I}^{(l)}(\mathbf{x})\right).$$

Let the data physical partition $\mathcal{D}^{(l)}$ be of the form

$$\mathcal{D}^{(l)} = \left\{D_j^{(l)}\right\}_{j=1}^{m_l}, \tag{3.12}$$

where m_l is the number of disjoint elements of the data physical partition $\mathcal{D}^{(l)}$ and each element $D_j^{(l)}$ is a subset of the input data set $\mathcal{D}$ such that the value of the network indicator function $\mathcal{I}^{(l)}$ is same for all points in $D_j^{(l)}$. Denote this value by $\mathcal{I}^{(l)}_{D_j^{(l)}}$, then we have

$$\mathcal{I}^{(l)}(\mathbf{x}) = \mathcal{I}^{(l)}_{D_j^{(l)}} \quad \text{for } \mathbf{x} \in D_j^{(l)}. \tag{3.13}$$

For each element $D \in \mathcal{D}^{(l)}$, denote the centroid of D by

$$\mathbf{x}_D = \frac{1}{|D|} \sum_{\mathbf{x}_i \in \mathcal{D}} w_i \mathbf{x}_i, \tag{3.14}$$

and the covariance matrix of D formed by vectors $\mathbf{x}_i - \mathbf{x}_D$ for all $\mathbf{x}_i \in D$ by

$$\mathrm{CoV}_D = \sum_{\mathbf{x}_i \in D} \left[\mathbf{x}_i - \mathbf{x}_D\right]^T \left[\mathbf{x}_i - \mathbf{x}_D\right], \tag{3.15}$$

where $\mathbf{x}_i - \mathbf{x}_D$ is a d-dimensional row vector. Then each element $D \in \mathcal{D}^{(l)}$ has d principal directions that correspond to the eigenvectors of CoV_D.

4. Adaptive Network Enhancement Method. This section describes the adaptive network enhancement method (ANE) for problem (1.1).

To this end, denote the current neural network, approximation, and error at the k^{th} adaptive step by

$$\mathcal{M}^{(k)}(l_k), \quad f^{(k)}(\mathbf{x}), \quad \text{and} \quad \xi^{(k)} = L\left(f^{(k)}\right),$$

respectively, where l_k is the number of hidden-layers of the network $\mathcal{M}^{(k)}(l_k)$. When accuracy of the current approximation $f^{(k)}(\mathbf{x})$ is not within the prescribed tolerance, i.e., $\xi^{(k)} > \epsilon\, L(0)$, the

network $\mathcal{M}^{(k)}(l_k)$ is enhanced by adding neurons at the either l_k-th or (l_k+1)-th hidden-layer. The latter means that we starts a new hidden-layer.

To determine the number of neurons to be added at the l_k-th hidden-layer, we use the *local network enhancement strategy* based on the data physical partition of $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^M$:

$$\mathcal{D}^{(l_k)}\left(f^{(k)}\right) = \mathcal{D} \cap \mathcal{K}^{(l_k)}\left(f^{(k)}\right) = \left\{\mathcal{D} \cap K : K \in \mathcal{K}^{(l_k)}\left(f^{(k)}\right)\right\}$$

by the current approximation $f^{(k)}(\mathbf{x})$. Specifically, we divide $\mathcal{D}^{(l_k)}\left(f^{(k)}\right)$ into two disjoint subsets,

$$\mathcal{D}^{(l_k)}\left(f^{(k)}\right) = \hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right) \bigcup \left(\mathcal{D}^{(l_k)}\left(f^{(k)}\right) \setminus \hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right)$$

where $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right) = \mathcal{D} \cap \hat{\mathcal{K}}^{(l_k)}\left(f^{(k)}\right)$ is a subset of $\mathcal{D}^{(l_k)}\left(f^{(k)}\right)$ consisting of elements in $\mathcal{D}^{(l_k)}\left(f^{(k)}\right)$ such that $f^{(k)}(\mathbf{x})$ is not yet a good approximation. Then the *enhancement strategy* is to add $\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right|$ new neurons to the l_k-th hidden-layer, where

$$\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right| = \text{ the number of elements of } \hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right). \tag{4.1}$$

To generate $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$, we employ the so-called marking strategy. There are two commonly used marking strategies for adaptive mesh refinement. One is the average marking strategy and the other is the bulk marking strategy. To describe these marking strategies, let us first introduce the following local error indicator

$$\xi_D^{(k)} = \left(\sum_{\mathbf{x}_i \in D} w_i \left(f^{(k)}(\mathbf{x}_i) - y_i\right)^2\right)^{1/2} \tag{4.2}$$

for each element $D \in \mathcal{D}^{(l_k)}\left(f^{(k)}\right)$. Clearly, we have

$$\xi^{(k)} = \left(\sum_{i=1}^M w_i \left(f^{(k)}(\mathbf{x}_i) - y_i\right)^2\right)^{1/2} = \left(\sum_{D \in \mathcal{D}^{(l_k)}\left(f^{(k)}\right)} \left(\xi_D^{(k)}\right)^2\right)^{1/2}. \tag{4.3}$$

The average marking strategy is given by

$$\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right) = \left\{D \in \mathcal{D}^{(l_k)}\left(f^{(k)}\right) : \xi_D^{(k)} \geq \frac{1}{\left|\mathcal{D}^{(l_k)}\left(f^{(k)}\right)\right|} \sum_{D \in \mathcal{D}^{(l_k)}\left(f^{(k)}\right)} \xi_D^{(k)}\right\}. \tag{4.4}$$

The bulk marking strategy is to find a minimal subset $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ such that

$$\sum_{D \in \hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)} \left(\xi_D^{(k)}\right)^2 \geq \gamma_1 \sum_{D \in \mathcal{D}^{(l_k)}\left(f^{(k)}\right)} \left(\xi_D^{(k)}\right)^2 \quad \text{for } \gamma_1 \in (0,\, 1). \tag{4.5}$$

The enhancement strategy adding $\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right|$ new neurons is suitable for all hidden-layers. Nevertheless, it may not be efficient for hidden-layers beyond the first hidden-layer. Notice that a multi-layer network is capable of generating piece-wise breaking hyper-planes in connected subdomains by one neuron. This observation motivates the notion of the *reduced number of elements* in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$. To this end, let

$$\hat{\mathcal{D}}^{(l)}\left(f^{(k)}\right) = \left\{\hat{D}_j^{(l)}\right\}_{j=1}^{\hat{m}_l}. \tag{4.6}$$

That is, there are $\hat{m}_l$ marked elements in $\mathcal{D}^{(l)}\left(f^{(k)}\right)$. Any two elements in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ are said to be disconnected if there is no pass connecting these two elements by elements of $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$. Let us group connected elements of $\hat{\mathcal{D}}^{(l)}\left(f^{(k)}\right)$ to form a set, whose elements are disconnected, denoted by

$$\tilde{\mathcal{D}}^{(l)}\left(f^{(k)}\right) = \left\{\tilde{D}_j^{(l)}\right\}_{j=1}^{\tilde{m}_l}, \tag{4.7}$$

where each element $\tilde{D}_j^{(l)} \in \tilde{\mathcal{D}}^{(l)}\left(f^{(k)}\right)$ is either an element of $\hat{\mathcal{D}}^{(l)}\left(f^{(k)}\right)$ or a union of connected elements in $\hat{\mathcal{D}}^{(l)}\left(f^{(k)}\right)$. Obviously, $\tilde{m}_l \leq \hat{m}_l$. Now, we define the reduced number of elements in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ by

$$\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right|_r = \begin{cases} \hat{m}_l, & l_k = 1, \\ \tilde{m}_l, & l_k \geq 2. \end{cases} \tag{4.8}$$

where any two elements in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ are disjoint if there is no pass connecting these two elements by elements of $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$.

REMARK 4.1. *For any two elements in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$, if values of their network indicator function differ only for one neuron, e.g., the i^{th} neuron at the k^{th} hidden-layer, then these two elements are neighbor and share part of the poly-hyper-plane $\mathcal{P}_i^{(k)}\left(\boldsymbol{\Theta}^{(k-1)}, \boldsymbol{\theta}_i^{(k)}\right)$ defined in (3.8).*

To address question (b) in section 1, i.e., when to add a new hidden-layer, we introduce a computable quantity, referred as the *improvement rate*, defined by

$$\eta_r^{(k)} = \left(\frac{\xi^{(k-1)} - \xi^{(k)}}{\xi^{(k-1)}}\right) \Big/ \left(\frac{\left(M^{(k)}(l_k)\right)^r - \left(M^{(k-1)}(l_{k-1})\right)^r}{\left(M^{(k)}(l_k)\right)^r}\right), \tag{4.9}$$

where $M^{(k-1)}(l_{k-1})$ and $M^{(k)}(l_k)$ denote the numbers of parameters of the networks $\mathcal{M}^{(k-1)}(l_{k-1})$ and $\mathcal{M}^{(k)}(l_k)$, respectively; and r is the order of the approximation with respect to the number of parameters and may depend on the activation function and the layer. The improvement rate measure a rate of improvement of two consecutive networks per the relative increase of parameters. If the improvement rate $\eta_r^{(k)}$ is less than or equal to a prescribed expectation rate $\delta \in (0, 2)$, i.e.,

$$\eta_r^{(k)} \leq \delta, \tag{4.10}$$

for two consecutive adaptive steps, then the ANE adds a new hidden-layer. Otherwise, the ANE adds neurons to the l_k-th hidden-layer of the current network $\mathcal{M}^{(k)}(l_k)$.

The ANE method for generating a nearly minimal multi-layer neural network is described in Algorithm 3.1.

5. Initialization of training. This section discusses initialization strategies of parameters of neural network in two dimensions. Extensions to three dimensions are straightforward.

The optimization problem in Step (7) of Algorithm 3.1 is non-convex and, hence, computationally intensive and complicated. Currently, this problem is often solved by either the first- or second-order iterative optimization methods such as gradient-based methods or Newton-like methods (see survey papers [2, 3] and references therein). Since non-convex optimizations usually have many solutions and/or many local minimums, it is then critical to start with a good initial guess in order to obtain the desired solution.

The ANE method itself is a natural continuation process for generating good initialization. That is, the approximation $f^{(k)}(\mathbf{x})$ of the previous network $\mathcal{M}^{(k)}(l_k)$ is in general a good approximation to $f^{(k+1)}(\mathbf{x})$ defined in Step (7) of Algorithm 3.1 for the enhanced network $\mathcal{M}^{(k+1)}(l_{k+1})$.

Algorithm 3.1 Adaptive Network Enhancement.
Given a data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^M$ with $\mathbf{x}_i \in \Omega = [-1,1]^d$, positive weights $\{w_i\}_{i=1}^M$, and a tolerance $\epsilon > 0$ for accuracy, starting with a one hidden-layer network $\mathcal{M}^{(0)}(l_0)$ with a small number of neurons, compute $f^{(0)} = \underset{v \in \mathcal{M}^{(0)}(l_0)}{\arg\min} L(v)$ by an iterative solver, then for $k = 0, 1, 2, \ldots$,

(1) use the network indicator function to determine the data physical partition $\mathcal{D}^{(l_k)}\left(f^{(k)}\right)$;
(2) for each $D \in \mathcal{D}^{(l_k)}\left(f^{(k)}\right)$, compute the local indicator $\xi_D^{(k)}$ in (4.2) and the estimator $\xi^{(k)}$ in (4.3);
(3) if $\xi < \epsilon$, then stop; otherwise, go to Step (4);
(4) use a marking strategy to form the subset $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ and calculate $\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right|_r$;
(5) for a prescribed expectation rate $\delta \in (0,2)$, if (4.10) holds for two consecutive steps, then set $l_{k+1} = l_k + 1$; otherwise, set $l_{k+1} = l_k$;
(6) form network $\mathcal{M}^{(k+1)}(l_{k+1})$ by adding $\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right|_r$ new neurons to the l_{k+1}-th hidden-layer;
(7) compute $f^{(k+1)} = \underset{v \in \mathcal{M}^{(k+1)}(l_{k+1})}{\arg\min} L(v)$ by an iterative solver.

Therefor, the trained nonlinear parameters of $\mathcal{M}^{(k)}(l_k)$ for $f^{(k)}(\mathbf{x})$ are good initials for the corresponding nonlinear parameters of the enhanced network $\mathcal{M}^{(k+1)}(l_{k+1})$. Based on this observation, below we discuss our initialization strategies for (1) parameters of the network $\mathcal{M}^{(0)}(l_0)$, (2) parameters of newly added neurons, and (3) linear (output) parameters of $f^{(k+1)}$.

Starting with a one hidden-layer network $\mathcal{M}^{(0)}(l_0)$ with relatively small number n_{l_0} of neurons, the approximation $f^{(0)}(\mathbf{x})$ has of the form

$$f^{(0)}(\mathbf{x}) = \mathbf{c}_1\, \sigma\left(\boldsymbol{\omega}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + c_0,$$

where $\boldsymbol{\Theta}^{(l_0)} = \left\{\boldsymbol{\theta}_i^{(1)}\right\}_{i=1}^{n_{l_0}} := \left\{\left(b_i^{(1)}, \boldsymbol{\omega}_i^{(1)}\right)\right\}_{i=1}^{n_1}$ are nonlinear parameters and $\mathbf{c} = (c_0, \mathbf{c}_1) \in \mathbb{R}^{n_{l_0}+1}$ are linear parameters. Initial of $\boldsymbol{\Theta}^{(l_0)}$ is chosen such that the hyper-lines

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

partition the domain $\Omega = (0,1)^2$ uniformly. Initial of $\mathbf{c}$ is set to be the solution of the system of linear algebraic equations

$$\boldsymbol{M}^{(l_0)}\left(\boldsymbol{\Theta}^{(l_0)}\right)\mathbf{c} = F^{(l_0)}\left(\boldsymbol{\Theta}^{(l_0)}\right) \tag{5.1}$$

defined in a similar fashion as (2.9).

Next, we discuss how to initialize the biases and weights of newly added neurons of the network $\mathcal{M}^{(k+1)}(l_{k+1})$. There are three cases:

$$(1)\ l_{k+1} = 1, \quad (2)\ l_{k+1} = l_k + 1, \quad \text{and} \quad (3)\ l_{k+1} = l_k \geq 2.$$

Case (1) means that the new neurons are added at the first hidden-layer. By associating each new neuron with an element $D \in \hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$, we initialize this neuron by setting its corresponding breaking line to pass through the centroid $\mathbf{x}_D$ and orthogonal to the principal direction that corresponds to the smallest eigenvalue of the covariance matrix CoV_D.

Consider Case (2). When $l_{k+1} = l_k + 1$, we start a new hidden-layer with $\left|\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)\right|_r$ neurons. By the definition in (4.8), we associate each neuron at the new hidden-layer $l_{k+1} = l_k + 1$ with an isolated element or an element consisting of several connected elements in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ and denote its bias and weights by

$$\boldsymbol{\theta}^{(l_{k+1})} = \left(b^{(l_{k+1})}, \boldsymbol{\omega}^{(l_{k+1})}\right) \in \mathbb{R}^{n_{l_k}+1} = \left(b^{(l_{k+1})}, \omega_1^{(l_{k+1})}, \cdots, \omega_{n_{l_k}}^{(l_{k+1})}\right) \in \mathbb{R}^{n_{l_k}+1}. \tag{5.2}$$

As section 2, denote the corresponding pre-activation function of the neuron by

$$g^{(l_{k+1})}(\mathbf{x}) = \boldsymbol{\omega}^{(l_{k+1})}\left(\mathbf{N}^{(l_k)} \circ \cdots \circ \mathbf{N}^{(1)}(\mathbf{x})\right) + b^{(l_{k+1})}$$

If the corresponding element D is an isolated element in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$, let $l_D(\mathbf{x}) = 0$ be the line that passes through the centroid $\mathbf{x}_D$ of D and is orthogonal to the direction vector with the lowest variance of D (see section 3). Denote by $\mathbf{x}_d$ the projection of a point in D onto the line $l_D(\mathbf{x}) = 0$ and whose distance to $\mathbf{x}_D$ is the largest among projections of all points in D onto the line. Then initial $\boldsymbol{\theta}_D^{(l_{k+1})}$ of the parameter $\boldsymbol{\theta}^{(l_{k+1})}$ is set to be

$$\boldsymbol{\theta}_D^{(l_{k+1})} = \underset{\boldsymbol{\theta}^{(l_{k+1})} \in \mathbb{R}^{n_{l_k}+1}}{\arg\min} \left\{\left(g^{(l_{k+1})}(\mathbf{x}_D)\right)^2 + \left(g^{(l_{k+1})}(\mathbf{x}_d)\right)^2\right\}. \tag{5.3}$$

When the corresponding element D consists of several connected elements in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$, denote the collection of these connected elements by $\mathcal{C}$. For each element $C \in \mathcal{C}$, denote by $\mathbf{x}_C$ the centroid of C. Then initial $\boldsymbol{\theta}_{\mathcal{C}}^{(l_{k+1})}$ of the parameter $\boldsymbol{\theta}^{(l_{k+1})}$ is set to be

$$\boldsymbol{\theta}_{\mathcal{C}}^{(l_{k+1})} = \underset{\boldsymbol{\theta}^{(l_{k+1})} \in \mathbb{R}^{n_{l_k}+1}}{\arg\min} \sum_{C \in \mathcal{C}} \left(g^{(l_{k+1})}(\mathbf{x}_C)\right)^2. \tag{5.4}$$

Now, let us consider Case (3) where new neurons are added at the current layer $l_{k+1} = l_k$. Let $s \in \{1, \ldots, k-1\}$ be the largest integer such that $l_{k-s} = l_k - 1$. Then $\mathcal{M}^{(k-s)}(l_{k-s})$ is the final network with $l_{k-s} = l_k - 1$ hidden-layers. Hence the weights and bias of each neuron associated with an element in $\hat{\mathcal{D}}^{(l_k)}\left(f^{(k)}\right)$ has the form of

$$\boldsymbol{\theta}^{(l_{k+1})} = \left(b^{(l_{k+1})}, \boldsymbol{\omega}^{(l_{k+1})}\right) = \left(b^{(l_{k+1})}, \omega_1^{(l_{k+1})}, \cdots, \omega_{n_{l_{k-s}}}^{(l_{k+1})}\right) \in \mathbb{R}^{n_{l_{k-s}}+1}. \tag{5.5}$$

Initial of $\boldsymbol{\theta}^{(l_{k+1})}$ in (5.5) can then be defined in a similar fashion as Case (2). Specifically, we have

$$\begin{cases} \boldsymbol{\theta}_D^{(l_{k+1})} = \underset{\boldsymbol{\theta}^{(l_{k+1})} \in \mathbb{R}^{n_{l_{k-s}}+1}}{\arg\min} \left\{\left(g^{(l_{k+1})}(\mathbf{x}_D)\right)^2 + \left(g^{(l_{k+1})}(\mathbf{x}_d)\right)^2\right\} \quad \text{and} \\ \boldsymbol{\theta}_{\mathcal{C}}^{(l_{k+1})} = \underset{\boldsymbol{\theta}^{(l_{k+1})} \in \mathbb{R}^{n_{l_{k-s}}+1}}{\arg\min} \sum_{C \in \mathcal{C}} \left(g^{(l_{k+1})}(\mathbf{x}_C)\right)^2, \end{cases} \tag{5.6}$$

where $\mathbf{x}_D$, $\mathbf{x}_d$, and $\mathbf{x}_C$ are defined in a similar way as in Case (2).

Finally, initial of the linear parameter $\mathbf{c} = (c_0, \mathbf{c}_1) \in R^{n_{l_{k+1}}+1}$ of $f^{k+1}(\mathbf{x})$ is set to be the solution of the system of algebraic linear equations

$$\boldsymbol{M}^{(l_{k+1})}\left(\boldsymbol{\Theta}^{(l_{k+1})}\right)\mathbf{c} = F^{(l_{k+1})}\left(\boldsymbol{\Theta}^{(l_{k+1})}\right) \tag{5.7}$$

defined in a similar fashion as (2.9).

6. Numerical Experiment. In this section, we report the numerical experiment on using the ANE method to approximate a function using the least-squares loss. The target function is defined on the domain $\Omega = [-1, 1]^2$, and is given by

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

For small constant α, this function exhibits a sharp transitional layer across a circular interface.

For this experiment, we set a small $\alpha = 0.01$ to test approximation accuracy using ANE. and the corresponding target function f is depicted in Fig. 2(a). A data set $\mathcal{D}$ for training network is generated using a fixed set 200×200 of quadrature points that are uniformly distributed in the domain Ω.

During the ANE process, we adopt the bulk marking strategy defined in (4.5) with $\gamma_1 = 0.5$ and choose the expectation rate $\delta = 0.6$ with $r = 1$ in (4.9); and the expected precision $\epsilon = 0.05$. The ANE method started with an initial network of 12 neurons in one hidden layer. The corresponding breaking lines $\{\mathcal{P}_i\}_{i=1}^{12}$ of these 12 neurons were uniformly initialized within the domain. Specifically, half of breaking lines are parallel to the x-axis

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

and the other half are parallel to the y-axis

$$\boldsymbol{\omega}_i^{(1)} = (1, 0) \quad \text{and} \quad b_i^{(1)} = -1 + \frac{1}{3}(i - 6) \quad \text{for} \quad i = 6, \cdots, 12.$$

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

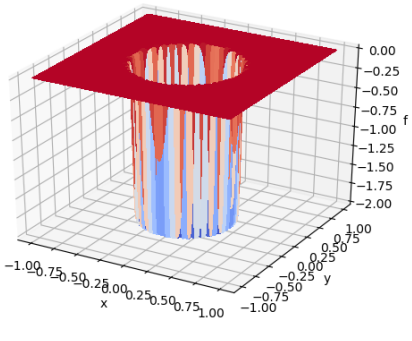
For each iteration of the ANE process, the corresponding minimization problem in (1.1) is solved iteratively using the Adam version of gradient descent [9] with a fixed learning rate 0.005. Adam's iterative solver is terminated when the relative change of the loss function $\|f - \hat{f}\|_\tau$ is less than 10^{-3} per 2000 iterations.

TABLE 1
Numerical results for using ANE to approximate function with a circular transitional layer

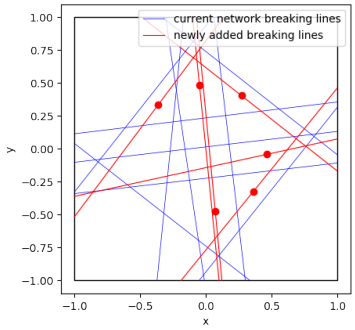
NN structure	# parameters	Approximation accuracy $\|f - \hat{f}\|_\tau / \|f\|$	Improvement rate η
2-12-1	37	0.357414	–
2-18-1	**55**	**0.323118**	**0.293198**
2-26-1	93	0.272614	0.382528
2-18-5-1	**137**	**0.025483**	**1.538967**

The ANE process is automatically terminated after four loops (see Table 1 for the intermediate and final result), and the final network model generated by the ANE is 2-18-5-1[1] with 137 parameters. The final network approximation model and the corresponding physical partition are shown in Figs. 2 (e) and (d). Using a relatively small set of parameters, ANE is able to accurately approximate a function with a thin transition layer without any oscillations. This remarkable approximation property can be explained by the fact that the circular interface of the underlying function is captured very effectively by a few breaking poly-lines generated in *the second hidden layer*, see the closed breaking lines formed by the 5 neurons in the second hidden layer in Fig. 2 (d).

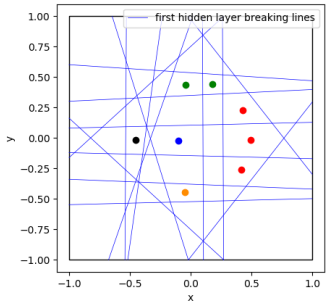
[1]The structure of a two- or three-hidden-layer network is expressed as 2-n_1-1 or 2-n_1-n_2-1, respectively, where n_1 and n_2 are the number of neurons at the first and second hidden-layer.



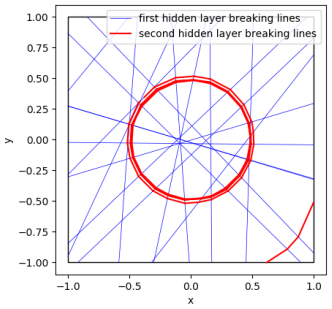
(a) The target function f with a circular transitional layer



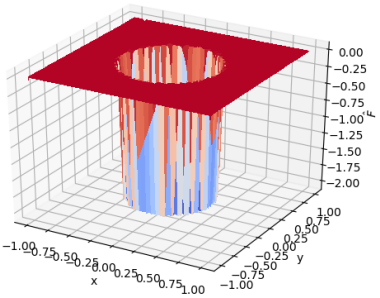
(b) PP of the 2-12-1 network and centroids of the marked elements (red dots)



(c) PP by 2-18-1 network and iso-lated and connected sub-domains (dots)



(d) PP by adaptive 2-18-5-1 net-work



(e) Approximation result using adaptive 2-18-5-1 network

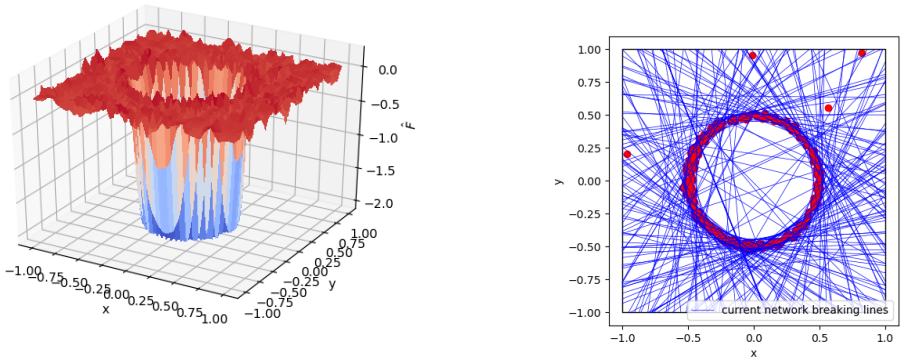
FIG. 2. *Adaptive approximation results for function with a transitional layer*

Figs. 2 (b)-(c) plot the physical partitions of the NN models at the intermediate adaptive process. In Fig. 2 (b), the centroids of the marked elements are illustrated by red dots; the breaking lines corresponding to the current and newly added neurons are shown by blue and red lines, respectively. Notice that the newly added neurons are initialized with break lines that pass through the centroids and align with the principal directions of the marked elements. Fig. 2(c) shows that there are 8 marked elements and 5 disjoint elements, which explains that 5 neurons are added to the second hidden layer during the neuron enhancement step.

TABLE 2
Numerical results of adaptive and fixed networks for function with a transitional layer

Network structure	# parameters	Approximation accuracy $\|f-\hat{f}\|_{\mathcal{T}}/\|f\|$
2-18-5-1 (Adaptive)	137	2.5483 %
2-18-5-1 (Fixed)	137	4.6199%
2-174-1 (Fixed)	523	11.1223%

For the purpose of a comparative study, we conducted function approximation experiments using two fixed network structures. As outlined in Table 2, when utilizing the same network structure (2-18-5-1), the resulting approximation accuracy is inferior to that achieved by the ANE



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

FIG. 3. *Approximation results generated by a fixed* 2-174-1 *network for function with a transitional layer*

method. The first two rows of Table 2 suggest that the ANE method provides a good initialization, which may simplify the non-convex optimization problem.

In the second experiment, we employed a fixed one- hidden-layer network (2-174-1) with nearly four times the number of parameters compared to the adaptive network. Despite the increased degrees of freedom and complexity, its approximation is less accurate (refer to the third row of Table 2). Furthermore, the approximated NN model exhibits a certain degree of oscillation (see Fig. 3 (a)), although the corresponding physical partition (Fig. 3(b)) still captures the narrow transition layer.

In general, a one-hidden-layer network necessitates dense breaking lines to approximate a circular interface, and oscillations along the interface can be attributed to the global basis functions generated from the first hidden layer. This experiment highlights that a deeper network, as illustrated by the two-hidden-layer network in this example, is more efficient in approximating a function with a thin nonlinear transition layer or interface. This experimental observation aligns with the theoretical findings presented in [7].

7. Conclusion. Designing an optimal deep neural network for a given task is important and challenging in many machine learning applications. This chapter provides a comprehensive introduction to the adaptive network enhancement (ANE) method, proposed recently in [11, 10, 5], which generates 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 that determine when a new hidden-layer and how many new neurons should be added when the current network is not sufficient for the task. This adaptive algorithm learns not only from given information (data, function, partial differential equation) 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, this chapter discusses an advanced procedure for initializing newly added neurons at the current or next hidden-layer.

Functions and partial differential equations with sharp transitions or discontinuities at unknown location have been computationally challenging, when approximated using other functional classes such as polynomials or piece-wise polynomials with fixed meshes. It was demonstrated

numerically in [11, 10, 5] that the ANE method can automatically design a nearly minimal two- or multi-hidden-layer network to learn functions exhibiting sharp transitional layers as well as continuous/discontinuous solutions of partial differential equations.

REFERENCES

[1] E. L. Allgower and K. Georg. *Numerical Continuation Methods: An Introduction*. Springer-Verlag, Berlin, 1990.

[2] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2018.

[3] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[4] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *Journal of Computational Physics*, 443 (2021) 110514.

[5] Z. Cai, J. Chen, and M. Liu. Self-adaptive deep neural network: Numerical approximation to functions and PDEs. *J. Comput. Phys.*, 455 (2022) 111021.

[6] Z. Cai, J. Chen, M. Liu, and X. Liu. Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic pdes. *J. Comput. Phys.*, 420 (2020) 109707.

[7] Z. Cai, J. Choi, and M. Liu. Least-squares neural network (LSNN) method for linear advection-reaction equation: general discontinuous interface. *arXiv:2301.06156v3[math.NA]*, 2023.

[8] Z. Cai, T. Ding, M. Liu, X. Liu, and J. Xia. A damped block gauss-newton method for shallow relu neural network. *manuscript*, 2023.

[9] D. P. Kingma and J. Ba. ADAM: A method for stochastic optimization. In *International Conference on Representation Learning, San Diego*, 2015; arXiv preprint arXiv:1412.6980.

[10] M. Liu and Z. Cai. Adaptive two-layer relu neural network: II. Ritz approximation to elliptic PDEs. *Comput. Math. Appl.*, 113:103–116, 2022.

[11] M. Liu, Z. Cai, and J. Chen. Adaptive two-layer ReLU neural network: I. best least-squares approximation. *Comput. Math. Appl.*, 113:34–44, 2022.