# Better Approximations of High Dimensional Smooth Functions by Deep Neural Networks with Rectified Power Units

Bo Li[b,a,1], Shanshan Tang[b,a,1], Haijun Yu[a,b,c,*]

[a] *LSEC, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Beijing 100190, China*
[b] *School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China*
[c] *National Center for Mathematics and Interdisciplinary Sciences, Chinese Academy of Sciences, Beijing 100190, China*

## Abstract

Deep neural networks with rectified linear units (ReLU) are getting more and more popular due to its universal representation power and successful applications. Some theoretical progresses on deep ReLU network approximations for functions in Sobolev space and Korobov space have recently been made by [D. Yarotsky, Neural Network, 94:103-114, 2017] and [H. Montanelli and Q. Du, SIAM J Math. Data Sci., 1:78-92, 2019]. Following similar approaches, we show that deep networks with rectified power units (RePU) can give better approximations for smooth functions than deep ReLU networks. Our analysis bases on classical polynomial approximation theory and some efficient algorithms proposed in this paper to convert polynomials into deep RePU networks of optimal size without any approximation error. Comparing to the results on ReLU network, the sizes of RePU networks required to approximate functions in Sobolev space and Korobov space with an error tolerance $\varepsilon$, by our constructive proofs, are in general $\mathcal{O}(\log \frac{1}{\varepsilon})$ times smaller than the sizes of corresponding ReLU networks. Our constructive proofs reveal the relation between the depth of the RePU network and the "order" of polynomial approximation. Taking into account some other good properties of RePU networks, such as being high-order differentiable and requiring less arithmetic operations, we advocate the use of deep RePU networks for problems where the underlying high dimensional functions are smooth or derivatives are involved in the loss function.

*Keywords:* deep neural network, high dimensional approximation, sparse grids, rectified linear unit, rectified power unit, rectified quadratic unit

## 1. Introduction

Artificial neural network, whose origin may date back to 1940s[1], is one of the most powerful tools in the field of machine learning. Especially, it became dominant in a lot of applications after the seminar works by Hinton et al.[2] and Bengio et al.[3] on efficient training of deep neural networks (DNNs), which pack up multi-layers of units with some nonlinear activation function. Since then, DNNs have greatly boosted the developments of image classification, speech recognition, computational chemistry and numerical solutions of high-dimensional partial differential equations, etc., see e.g. [4][5][6][7][8] to name a few.

The success of DNNs relies on two facts: 1) DNN is a powerful tool for general function approximation; 2) Efficient training methods are available to find minimizers with good generalization ability. In this paper, we focus on the first fact. It is known that artificial neural networks can approximate any $C^0$ and $L^1$ functions with any given error tolerance, using only one hidden layer (see e.g. [9][10][11]). However, people have realized recently that deep networks have better representation power[12][13][14]. One of the commonly used activation functions with DNN is the so called rectified linear unit (ReLU)[15], which is defined as

[*] Corresponding author. Email: hyu@lsec.cc.ac.cn

[1] Contributed equally. Author list is alphabetical. Email:libo1171309228@lsec.cc.ac.cn, tangshanshan@lsec.cc.ac.cn

$\sigma(x) = \max(0, x)$. Telgarsky [13] gave a simple and elegant construction showing that for any $k$, there exist $k$-layer, $\mathcal{O}(1)$ wide ReLU networks on one-dimensional data, which can express a sawtooth function on $[0,1]$ with $\mathcal{O}(2^k)$ oscillations. Moreover, such a rapidly oscillating function cannot be approximated by poly($k$)-wide ReLU networks with $o(k/\log(k))$ depth. Following this approach, several other works proved that deep ReLU networks have better approximation than shallow ReLU networks [16][17][18][19]. In particular, for $C^\beta$-differentiable $d$-dimensional functions, Yarotsky [18] proved that the number of parameters needed to achieve an error tolerance of $\varepsilon$ is $\mathcal{O}(\varepsilon^{-\frac{d}{\beta}} \log \frac{1}{\varepsilon})$. Petersen and Voigtlaender [19] proved that for a class of $d$-dimensional piecewise $C^\beta$ continuous functions with the discontinuous interfaces being $C^\beta$ continuous also, one can construct a ReLU neural network with $\mathcal{O}((1 + \frac{\beta}{d}) \log_2(2 + \beta))$ layers, $\mathcal{O}(\varepsilon^{-\frac{2(d-1)}{\beta}})$ nonzero weights to achieve $\varepsilon$-approximation. The complexity bound is sharp. For analytic functions, E and Wang [20] proved that using ReLU networks with fixed width $d + 4$, to achieve an error tolerance of $\varepsilon$, the depth of the network depends on $\log \frac{1}{\varepsilon}$ instead of $\varepsilon$ itself. Note that there is also an exponential dependence on the dimension $d$.

One basic fact on ReLU networks is that function $x^2$ can be approximated within any error $\varepsilon > 0$ by a ReLU network having the depth, the number of weights and computation units all of order $\mathcal{O}(\log \frac{1}{\varepsilon})$. This fact has been used by several groups (see e.g. [16][18]) to analyze the approximation property of general smooth functions using ReLU networks. In this paper, we extend the analysis to deep neural networks using rectified power units (RePUs), which are defined as

$$\sigma_s(x) = \begin{cases} x^s, & x \geq 0, \\ 0, & x < 0, \end{cases}, \quad s \in \mathbb{N}, \tag{1.1}$$

where $\mathbb{N}$ denotes the set of all positive integers. Note that $\sigma_1$ is the commonly used ReLU function. We call $\sigma_2$, $\sigma_3$ rectified quadratic unit (ReQU) and rectified cubic unit (ReCU), respectively. We show that deep neural networks using RePUs($s \geq 2$) as activation functions have better approximation property for smooth functions than those using ReLUs. By replacing ReLU with RePU, the functions $x$, $x^2$ and $xy$ can be exactly represented with no approximation error using networks having just a few nodes and nonzero weights. Based on this, we build an efficient algorithm to explicitly convert any function from a polynomial space into a RePU network having approximately same number of coefficients. This allows us to obtain a better upper bound of the best neural network approximation for general smooth functions using classical polynomial approximation theories.

For high dimensional problems, to be tractable, the intrinsic dimension usually do not grow as fast as the observation dimension. In other words, the problems have low dimensional structure. A particular example is the high-dimensional smooth functions with bounded mixed derivatives, for which sparse grid (or hyperbolic cross) approximation is a very popular approximation tool [21][22][23][24]. In the past few decades, sparse grid method and hyperbolic cross approximations have been applied to many applications, for example, numerical integration and interpolation[21][25][26],[27], solving partial differential equations (PDEs)[28][29][30][31][32], computational chemistry[23][33][34][35], uncertainty quantification[36][37][38], etc. Recently, the connection between linear finite element sparse grids and deep ReLU neural networks has been used by Montanelli and Du [39] to obtain an upper bound of deep ReLU network approximation of high dimensional functions with bounded mixed derivatives. The relations between deep ReLU networks and general linear finite elements have also been studied by He et al.[40]. We use a similar but different approach. In our approach, we approximate multivariate functions in high order Korobov space using sparse grid Chebyshev interpolation [26] for the interpolation problem, and using hyperbolic cross spectral approximation for the projection problem [24][29]. And then convert the high-dimensional polynomial into a ReQU network, instead of a ReLU network, to avoid adding an extra factor $\log \frac{1}{\varepsilon}$ in the size of the neural network. We find that RePU networks have the following good properties:

- The RePU neural networks provide better approximations for smooth functions comparing to ReLU neural network approximations. To achieve same accuracy, the RePU network approximation needs less number of layers and smaller network size. For example, for any analytic function, we can construct a ReQU network with no more than $\mathcal{O}\left(\log_2\left(\log \frac{1}{\varepsilon}\right)\right)$ layers, and no more than $\mathcal{O}\left(\frac{1}{\gamma} \log\left(\frac{1}{\varepsilon}\right)\right)$ nonzero weights to approximate it with error $\varepsilon$, where $\gamma = \mathcal{O}\left(\log(\log \frac{1}{\varepsilon})\right)$. More results are given in Theorem 4, 8, 10 (cp. Yarotsky [18], E and Wang [20], Petersen and Voigtlaender [19], Montanelli and Du [39]).

- The functions represented by RePU networks are smooth functions, so they naturally fit in the places where derivatives are involved in the loss function.
- Compared to other high-order differentiable activation functions, such as logistic, tanh, softplus, sinc etc., RePUs are more efficient in terms of number of arithmetic operations needed to evaluate, especially the rectified quadratic unit.

Based on the facts above, we advocate the use of deep RePU networks in places where the functions to be approximated are smooth.

The remaining part of this paper is organized as follows. In Section 2, we first show how to approximate univariate smooth functions using RePU networks by converting best polynomial approximations into RePU networks. Then we use a similar approach to analyze the ReQU network approximation for multivariate functions in weighted Sobolev space in Section 3. After that, we show how high-dimensional functions with sparse polynomial approximations can be well approximated by ReQU networks in Section 4. We end the paper by a short summary in Section 5.

## 2. Approximation of univariate smooth functions by deep RePU networks

We first introduce some symbols and notations related to neural networks. Denote by $\mathbb{N}$ the set of all positive integers, $\mathbb{N}_0 := \{0\} \cup \mathbb{N}$. Let $d, L \in \mathbb{N}$, we denote a neural network $\Phi$ with input of dimension $d$, number of layer $L$, by a matrix-vector sequence

$$\Phi = \big((A_1, b_1), \cdots, (A_L, b_L)\big), \tag{2.1}$$

where $N_0 = d$, $N_1, \cdots, N_L \in \mathbb{N}$, $A_k$ are $N_k \times N_{k-1}$ matrices, and $b_k \in \mathbb{R}^{N_k}$.

If $\Phi$ is a neural network, and $\rho : \mathbb{R} \to \mathbb{R}$ is an arbitrary activation function, then define

$$R_\rho(\Phi) : \mathbb{R}^d \to \mathbb{R}^{N_L}, \qquad R_\rho(\Phi)(\boldsymbol{x}) = \boldsymbol{x}_L, \tag{2.2}$$

where $R_\rho(\Phi)(\boldsymbol{x})$ is defined as

$$\begin{cases} \boldsymbol{x}_0 := \boldsymbol{x}, \\ \boldsymbol{x}_k := \rho(A_k \boldsymbol{x}_{k-1} + b_k), \quad k = 1, 2, \ldots, L-1, \\ \boldsymbol{x}_L := A_L \boldsymbol{x}_{L-1} + b_L, \end{cases} \tag{2.3}$$

and

$$\rho(\boldsymbol{y}) := \big(\rho(y^1), \cdots, \rho(y^m)\big), \quad \forall\ \boldsymbol{y} = (y^1, \cdots, y^m) \in \mathbb{R}^m.$$

We use three quantities to account the complexity of the neural network: number of hidden layers, number of nodes(i.e. activation units), and number of nonzero weights, which are $L-1$, $\sum_{k=1}^{L-1} N_k$ and number of non-zeros in $\{(A_k, b_k), k = 1, \ldots, L\}$, respectively, for the neural network defined in (2.1). For convenience, we denote by $\#A$ the number of nonzero components in $A$ for a given matrix or vector $A$. For the neural network $\Phi$ defined in (2.1), we also denote its number of nonzero weights as $\#\Phi := \sum_{k=1}^{L} (\#A_k + \#b_k)$.

In this paper we study the approximation property of smooth functions by deep neural networks with RePUs as activation units. Note that RePU $\sigma_s$ is a special case of piece-wise polynomial activation function, which has been studied in [11, 41] for shallow network approximation. We also note that $\sigma_3$ has been used in a deep Ritz method proposed to solve the variational problems related to PDEs [42].

### *2.1. Approximation by deep ReQU networks*

Our analyses rely upon the fact: $x$, $x^2$, $\ldots$, $x^s$, and $xy$ all can be realized by $\sigma_s$ neural networks with a few number of coefficients. We first give the result for $s = 2$ case.

**Lemma 1.** *For $\forall x, y \in \mathbb{R}$ the following identities hold:*

$$x^2 = \beta_2^T \sigma_2(\omega_2 x), \tag{2.4}$$

$$x = \beta_1^T \sigma_2(\omega_1 x + \gamma_1), \tag{2.5}$$

$$xy = \beta_1^T \sigma_2(\omega_1 x + \gamma_1 y), \tag{2.6}$$

*where*

$$\beta_1 = \frac{1}{4}[1, 1, -1, -1]^T, \quad \beta_2 = [1, 1]^T, \quad \omega_1 = [1, -1, 1, -1]^T, \quad \omega_2 = [1, -1]^T, \quad \gamma_1 = [1, -1, -1, 1]^T. \tag{2.7}$$

*If both $x$ and $y$ are non-negative, the formula for $x^2$ and $xy$ can be simplified to the following form*

$$x^2 = \sigma_2(x), \tag{2.8}$$

$$xy = \beta_3^T \sigma_2(\omega_3 x + \gamma_2 y), \tag{2.9}$$

*where*

$$\beta_3 = \frac{1}{4}[1, -1, -1]^T, \quad \omega_3 = [1, 1, -1]^T, \quad \gamma_2 = [1, -1, 1]^T. \tag{2.10}$$

*Proof.* All the identities can be obtained by straightforward calculations. □

Note that the realizations given in Lemma 1 are not unique. For example, to realize $id_X(x) = x$, we may use

$$x = (x + 1/2)^2 - x^2 - 1/4 = \beta_2^T \sigma_2(\omega_2(x + 1/2) - \beta_2^T \sigma_2(\omega_2 x) - 1/4,$$

for general $x \in \mathbb{R}$, and use

$$x = (x + 1/2)^2 - x^2 - 1/4 = \sigma_2((x + 1/2) - \sigma_2(x) - 1/4,$$

for non-negative $x$. To have a neat presentation, we will use (2.4)-(2.10) throughout this paper even though simpler realizations may exist for some special cases. We notice that realization of identity map $id_X(x)$ given in 2.5 is a special case of (2.6) with $y = 1$. And the constant function 1 can be represented by a trivial network with $L = 1$ and $A_1 = 0, b_1 = 1$ .

**Remark 1.** *Notice that in [18, 19, 39], all the analyses base on the fact that $x^2$ can be approximated to an error tolerance $\varepsilon$ by a ReLU deep networks of complexity $\mathcal{O}(\log \frac{1}{\varepsilon})$. In our approach, by replacing ReLU with ReQU, $x^2$ is represented with no error using a ReQU network with only one hidden layer and 2 activation functions.*

#### *2.1.1. Optimal realizations of polynomials by deep ReQU networks with no error*

The basic property of $\sigma_2$ given in Lemma 1 can be used to construct deep neural network representations of monomials and polynomials. We first show that the monomial $x^n, n > 2$ can be represented exactly by deep ReQU networks of finite size but not shallow ReQU networks.

**Theorem 1.** *A) The monomial $x^n, n \in \mathbb{N}$ defined on $\mathbb{R}$ can be represented exactly by a $\sigma_2$ network. The number of network layers, number of nodes and number of weights required to realize $x^n$ are at most $\lfloor \log_2 n \rfloor + 2$, $5\lfloor \log_2 n \rfloor + 5$ and $25\lfloor \log_2 n \rfloor + 14$, respectively. Here $\lfloor x \rfloor$ represents the largest integer not exceeding $x$ for $x \in \mathbb{R}$.*

*B) For any $n > 2$, $x^n$ can not be represented exactly by any ReQU network with only one hidden layer.*

*Proof.* We first prove part B. For a one-layer ReQU network with $N$ activation units, one input and one output, the function it presented can be written as

$$f_N(x) = \sum_{k=1}^{N} c_k \sigma_2(a_k x + b_k) + d,$$

where $d$ and $a_k, b_k, c_k, k = 1, \dots, N$ are the parameters of the network. Obviously, $f_N$ is a piecewise polynomial, with at most $N+1$ pieces in the intervals divided by distinct points of $x_k = -b_k/a_k, k = 1, \dots, N$(suppose the points are in ascending order). In each piece, $f_N$ is a polynomials of degree 2, so it can't represent $x^n, n > 2$ exactly. The error decreases at most cubicly with respect the length of the interval. So, to approximate $x^n, n > 2$ on a finite interval, e.g. $I = [-1, 1]$ with $N$ ReQU units, one can only obtain an algebraic convergence with respect to $N$.

Now we prove part A. We first express $n$ in binary system as follows:

$$n = a_m \cdot 2^m + a_{m-1} \cdot 2^{m-1} + \cdots + a_1 \cdot 2 + a_0,$$

where $a_j \in \{0, 1\}$ for $j = 0, 1, ..., m-1$, $a_m = 1$, and $m = \lfloor \log_2 n \rfloor$. Then

$$x^n = x^{2^m} \cdot x^{\sum\limits_{j=0}^{m-1} a_j 2^j}.$$

Introducing intermediate variables

$$\xi_k^{(1)} := x^{2^k}, \qquad \xi_k^{(2)} := x^{\sum\limits_{j=0}^{k-1} a_j 2^j}, \qquad \text{for } 1 \le k \le m,$$

then

$$x^n = \xi_m^{(1)} \xi_m^{(2)}. \tag{2.11}$$

We use the iteration scheme

$$\begin{cases} \xi_1^{(1)} = x^2, \\ \xi_1^{(2)} = x^{a_0}, \end{cases} \quad \text{and} \quad \begin{cases} \xi_k^{(1)} = (\xi_{k-1}^{(1)})^2, \\ \xi_k^{(2)} = (\xi_{k-1}^{(1)})^{a_{k-1}} \xi_{k-1}^{(2)}, \end{cases} \quad \text{for } 2 \le k \le m, \tag{2.12}$$

and (2.11) to realize $x^n$. The outline of the realization is demonstrated in Figure 1. In each iteration step, we need to realize two basic operations: $(x)^2$ and $(x)^{a_j} y$, where $x, y$ stands for $\xi_k^{(1)}, \xi_k^{(2)}$ respectively. Note that $(x)^2$ can be realized by equation (2.4) and (2.8) in Lemma 1. For operation $(x)^{a_j} y$, since $a_j \in \{0, 1\}$, by (2.6), we have

$$x^{a_j} y = \left( \frac{1 + (-1)^{a_j}}{2} + \frac{1 - (-1)^{a_j}}{2} x \right) y = \beta_1^T \sigma_2 \left( \omega_1 (c_j^+ + c_j^- x) + \gamma_1 y \right), \tag{2.13}$$

where $c_j^{\pm} := \frac{1 \pm (-1)^{a_j}}{2}$. So $x^{a_j} y$ can be realized by a linear combination of four $\sigma_2$ units.

Now we show the procedure in details. Obviously, a linear function $ax + b$ can be realized by a trivial one-layer network with no activation units. A quadratic polynomial $ax^2 + bx + c$ can be realized, using the representation $x(ax + b) + c = \beta_1^T \sigma_2(\omega_1(ax + b) + \gamma_1 x) + c$, by a ReLU network with one hidden layer, 4 activation units and 13 nonzero weights. For $n \ge 3$, we follow the idea given in equation (2.12) and Figure 1. The function $x^n$ are realized in $m + 1$ steps, which are discussed below.

1) In Step 1, we calculate

$$\xi_1^{(1)} = x^2 = \beta_2^T \sigma_2(\omega_2 x) \ge 0, \tag{2.14}$$

$$\xi_1^{(2)} = x^{a_0} = c_0^+ + c_0^- x = c_0^+ + c_0^- \beta_1^T \sigma_2 \left( \omega_1 x + \gamma_1 \right), \tag{2.15}$$

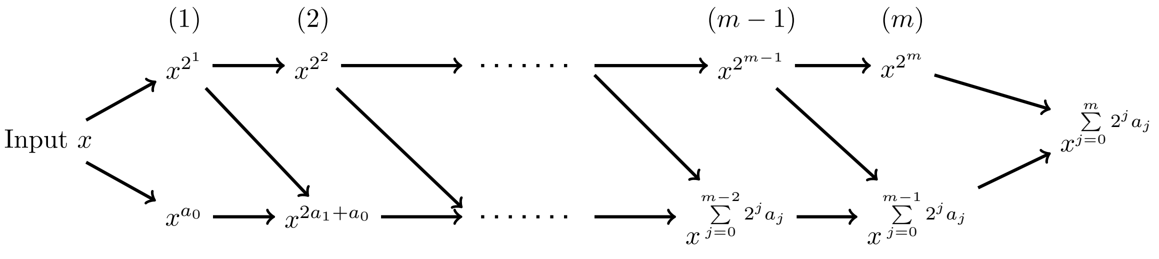

Figure 1: A schematic diagram for $\sigma_2$ network realization of $x^n$. $(j)$ represents the $j$-th hidden layer for intermediate variables.

which implies the first layer output of the neural network is:

$$\boldsymbol{x}_1 = \sigma_2(A_1 x + b_1), \quad \text{where} \quad A_1 = \begin{bmatrix} \omega_2 \\ \omega_1 \end{bmatrix}_{6\times 1}, \quad b_1 = \begin{bmatrix} \mathbf{0} \\ \gamma_1 \end{bmatrix}_{6\times 1}, \tag{2.16}$$

and

$$\begin{bmatrix} \xi_1^{(1)} \\ \xi_1^{(2)} \end{bmatrix} = A_{20}\boldsymbol{x}_1 + b_{20}, \quad \text{where} \quad A_{20} = \begin{bmatrix} \beta_2^T & \mathbf{0} \\ \mathbf{0} & c_0^- \beta_1^T \end{bmatrix}_{2\times 6}, \quad b_{20} = \begin{bmatrix} 0 \\ c_0^+ \end{bmatrix}_{2\times 1}. \tag{2.17}$$

Since $\#\omega_1 = 4$, $\#\omega_2 = 2$, $\#\gamma_1 = 4$, it is easy to see that the number of nodes in the first hidden layer is 6, and the number of non-zeros is: $\#A_1 + \#b_1 = 10$.

2) In Step $j$, $2 \le j \le m$, we calculate

$$\begin{aligned} \xi_j^{(1)} &= (\xi_{j-1}^{(1)})^2 = \sigma_2(\xi_{j-1}^{(1)}) \ge 0, & (2.18) \\ \xi_j^{(2)} &= (\xi_{j-1}^{(1)})^{a_{j-1}} \xi_{j-1}^{(2)} = (c_{j-1}^+ + c_{j-1}^- \xi_{j-1}^{(1)}) \xi_{j-1}^{(2)} \\ &= \beta_1^T \sigma_2 \left( \omega_1 (c_{j-1}^+ + c_{j-1}^- \xi_{j-1}^{(1)}) + \gamma_1 \xi_{j-1}^{(2)} \right), & (2.19) \end{aligned}$$

which suggest the $j$-th layer output of the neural network is:

$$\boldsymbol{x}_j = \sigma_2 \left( A_{j1} \begin{bmatrix} \xi_{j-1}^{(1)} \\ \xi_{j-1}^{(2)} \end{bmatrix} + b_{j1} \right), \quad \text{where } A_{j1} = \begin{bmatrix} 1 & 0 \\ c_{j-1}^- \omega_1 & \gamma_1 \end{bmatrix}_{5\times 2}, \ b_{j1} = \begin{bmatrix} 0 \\ c_{j-1}^+ \omega_1 \end{bmatrix}_{5\times 1},$$

and

$$\begin{bmatrix} \xi_j^{(1)} \\ \xi_j^{(2)} \end{bmatrix} = A_{j+1,0}\boldsymbol{x}_j + b_{j+1,0}, \quad \text{where } A_{j+1,0} = \begin{bmatrix} 1 & \mathbf{0} \\ 0 & \beta_1^T \end{bmatrix}_{2\times 5}, \ b_{j+1,0} = \mathbf{0}. \tag{2.20}$$

We have

$$A_j = A_{j1}A_{j0}, \quad b_j = A_{j1}b_{j0} + b_{j1}, \quad j = 2, \dots, m. \tag{2.21}$$

By a direct calculation, we find that the number of nodes in Layer $j$ is 5, and the number of non-zeros in Layer $j$, $j = 3, \dots, m$ is $\#A_j + \#b_j = 21 + 4 = 25$. For $j = 2$, $\#A_2 + \#b_2 = 26 + 4 = 30$.

3) In Step $m+1$, we calculate

$$x^n = \xi_m^{(1)} \xi_m^{(2)} = \beta_1^T \sigma_2 \left( \omega_1 \xi_m^{(1)} + \gamma_1 \xi_m^{(2)} \right), \tag{2.22}$$

which implies

$$\boldsymbol{x}_{m+1} = \sigma_2 \left( A_{m+1,1} \begin{bmatrix} \xi_m^{(1)} \\ \xi_m^{(2)} \end{bmatrix} \right), \quad \text{where} \quad A_{m+1,1} = [\omega_1 \ \gamma_1]_{4\times 2}. \tag{2.23}$$

So we get

$$A_{m+1} = A_{m+1,1} A_{m+1,0}, \quad b_{m+1} = \mathbf{0}, \tag{2.24}$$

and

$$\boldsymbol{x}_{m+2} := x^n = \beta_1^T \boldsymbol{x}_{m+1}. \tag{2.25}$$

By a direct calculation, we get the number of nodes in Layer $m+1$ is 4, and the number of non-zeros is $\#A_{m+1} = 20$.

For Layer $m+2$, which is the output layer of the overall network, $A_{m+2} = \beta_1^T$, and $b_{m+2} = 0$. There is no activation units and number of non-zeros is $\#A_{m+2} = 4$.

The ReQU network we just built has $m+2$ layers, number of nodes $6 + 5(m-1) + 4 = 5m + 5$, number of nonzero weights $10 + 30 + 25(m-2) + 20 + 4 = 25m + 14$. Combining the cases $n = 1, 2$, we reach to the desired conclusion. □

Now we consider how to convert univariate polynomials into $\sigma_2$ networks. If we directly apply Theorem 1 to each monomial term in a polynomial and then combine them together, one would obtain a network of depth $\mathcal{O}(\log_2 n)$ and size $\mathcal{O}(n \log_2 n)$, which is not optimal. We provide here two algorithms to convert a polynomial into a ReQU network of same scale, i.e. without the extra $\log_2 n$ factor. The first one is a direct implementation of Horner's method (also known as Qin Jiushao's algorithm in China):

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n \\ &= a_0 + x\Big(a_1 + x\big(a_2 + x(a_3 + \ldots + x(a_{n-1} + x a_n))\big)\Big). \end{aligned} \tag{2.26}$$

To describe the algorithm iteratively, we introduce the following intermediate variables

$$y_k = \begin{cases} a_{n-1} + x a_n, & k = n, \\ a_{k-1} + x y_{k+1}, & k = n-1, n-2, \ldots, 1. \end{cases}$$

Then we have $y_0 = f(x)$. But implementing of $y_k$ for each $k$, using realizations formula given in Lemma 1, and stack the implementations of $n$ steps up, we obtain a $\sigma_2$ neural network with $\mathcal{O}(n)$ layers and each layer has a constant width independent of $n$.

The second construction given in the following theorem can achieve same representation power with same amount of weights but less layers.

**Theorem 2.** *If $f(x)$ is a polynomial of degree $n$ on $\mathbb{R}$ , then it can be represented exactly by a $\sigma_2$ neural network with $\lfloor \log_2 n \rfloor + 1$ hidden layers, and number of nodes and nonzero weights are both of order $\mathcal{O}(n)$. To be more precise, the number of nodes is bounded by $9n$, and number of nonzero weights is bounded by $61n$.*

*Proof.* Assume $f(x) = \sum\limits_{j=0}^{n} a_j x^j$, $a_n \neq 0$. We first use an example with $n = 15$ to demonstrate the process of network construction as follows:

$$\begin{aligned}
f(x) &= a_{15}x^{15} + a_{14}x^{14} + \cdots + a_8x^8 + a_7x^7 + a_6x^6 + \cdots + a_1x + a_0 \\
&= \underbrace{x^8}_{\xi_{3,0}} \underbrace{\left\{ \underbrace{x^4}_{\xi_{2,0}} \underbrace{\left[ \underbrace{x^2}_{\xi_{1,0}} \underbrace{(a_{15}x + a_{14})}_{\xi_{1,8}} + \underbrace{(a_{13}x + a_{12})}_{\xi_{1,7}} \right]}_{\xi_{2,4}} + \underbrace{\left[ x^2 \underbrace{(a_{11}x + a_{10})}_{\xi_{1,6}} + \underbrace{(a_9x + a_8)}_{\xi_{1,5}} \right]}_{\xi_{2,3}} \right\}}_{\xi_{3,2}} \\
&\quad + \underbrace{\left\{ x^4 \underbrace{\left[ x^2 \underbrace{(a_7x + a_6)}_{\xi_{1,4}} + \underbrace{(a_5x + a_4)}_{\xi_{1,3}} \right]}_{\xi_{2,2}} + \underbrace{\left[ x^2 \underbrace{(a_3x + a_2)}_{\xi_{1,2}} + \underbrace{(a_1x + a_0)}_{\xi_{1,1}} \right]}_{\xi_{2,1}} \right\}}_{\xi_{3,1}}.
\end{aligned} \tag{2.27}$$

Here $\xi_{1,j_1}, j_1 = 0, 1, 2, \cdots, 8$, $\xi_{2,j_2}, j_2 = 0, 1, 2, \cdots, 4$, and $\xi_{3,j_3}$, $j_3 = 0, 1, 2$ are the intermediate variable output of Layer 1, 2, 3, respectively. And the final output is $f(x) = \xi_{3,0}\xi_{3,2} + \xi_{3,1}$.

We first describe the construction for the case $n \geq 4$ here.

Denote $m = \lfloor \log_2 n \rfloor$. We first extend $f(x)$ to include monomials up to degree $2^{m+1} - 1$ by zero padding:

$$f(x) := \sum_{j=0}^{2^{m+1}-1} a_j x^j, \qquad \text{where} \quad a_j = 0, \quad \text{for } n + 1 \leq j \leq 2^{m+1} - 1. \tag{2.28}$$

The process of building a $\sigma_2$ network to represent $f(x)$ is similar to the case $n = 15$. We give details below.

1) The output of Layer 1 intermediate variables are:

$$\xi_{1,j} = a_{2j-1}x + a_{2j-2} = a_{2j-1}\beta_1^T \sigma_2(\omega_1 x + \gamma_1) + a_{2j-2}, \quad j = 1, 2, ..., 2^m, \tag{2.29}$$

$$\xi_{1,0} = x^2 = \beta_2^T \sigma_2(\omega_2 x), \tag{2.30}$$

which suggest

$$\boldsymbol{x}_1 = \sigma_2 \begin{pmatrix} \omega_1 x + \gamma_1 \\ \omega_2 \end{pmatrix} = \sigma_2(A_1 x + b_1), \quad \text{where} \quad A_1 = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \quad b_1 = \begin{bmatrix} \gamma_1 \\ \mathbf{0} \end{bmatrix}. \tag{2.31}$$

and

$$\boldsymbol{\xi}_1 = A_{20}\boldsymbol{x}_1 + b_{20}, \quad \text{where} \quad A_{20} = \begin{bmatrix} a_{21}\beta_1^T & \mathbf{0} \\ \mathbf{0} & \beta_2^T \end{bmatrix}, \quad b_{20} = \begin{bmatrix} a_{22} \\ 0 \end{bmatrix}, \tag{2.32}$$

with $\boldsymbol{\xi}_1 = [\xi_{1,1}, \xi_{1,2}, \ldots, \xi_{1,2^m}, \xi_{1,0}]^T$, $a_{21} = [a_1, a_3, \ldots, a_{2^{m+1}-1}]^T$, $a_{22} = [a_0, a_2, \ldots, a_{2^{m+1}-2}]^T$.

2) The output of Layer 2 intermediate variables are:

$$\begin{aligned}
\xi_{2,j} &= \xi_{1,0}\xi_{1,2j} + \xi_{1,2j-1} \\
&= \beta_1^T \sigma_2(\omega_1 \xi_{1,2j} + \gamma_1 \xi_{1,0}) + \beta_1^T \sigma_2(\omega_1 \xi_{1,2j-1} + \gamma_1), \quad j = 1, 2, ..., 2^{m-1},
\end{aligned} \tag{2.33}$$

$$\xi_{2,0} = (\xi_{1,0})^2 = \sigma_2(\xi_{1,0}), \tag{2.34}$$

which imply

$$\boldsymbol{x}_2 = \sigma_2(A_{21}\boldsymbol{\xi_1} + b_{21}), \quad \boldsymbol{x_2}, b_{21} \in \mathbb{R}^{(8\times 2^{m-1}+1)\times 1}, \; A_{21} \in \mathbb{R}^{(8\times 2^{m-1}+1)\times(2^m+1)}, \tag{2.35}$$

and most elements in $A_{21}, b_{21}$ are zeros. The nonzero elements are given below using a Matlab subscript style as:

$$A_{21}(8(j-1)+1:8j,[2j-1:2j,2^m+1]) = \begin{bmatrix} \omega_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \omega_1 & \gamma_1 \end{bmatrix}, \quad b_{21}(8(j-1):8j) = \begin{bmatrix} \gamma_1 \\ \mathbf{0} \end{bmatrix}, \tag{2.36}$$

for $j = 1, 2, \ldots, 2^{m-1}$, and the last element of $A_{21}$ is 1. According to the result (2.32) of Layer 1, we get

$$\boldsymbol{x}_2 = \sigma_2\left(A_2\boldsymbol{x}_1 + b_2\right), \quad A_2 = A_{21}A_{20}, \quad b_2 = A_{21}b_{20} + b_{21}. \tag{2.37}$$

We also have

$$\boldsymbol{\xi}_2 = A_{30}\boldsymbol{x}_2, \quad \text{where} \quad A_{30} = \begin{bmatrix} I_{2^{m-1}} \otimes [\beta_1^T \ \beta_1^T] & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.38}$$

Here $\boldsymbol{\xi}_2 = [\xi_{2,1}, \xi_{2,2}, \ldots, \xi_{2,2^{m-1}}, \xi_{2,0}]^T$, and $I_{2^{m-1}}$ is the identity matrix of in $\mathbb{R}^{2^{m-1}}$. $\otimes$ stands for Kronecker product.

3) The output of Layer $k$ ($3 \leq k \leq m$) intermediate variables are:

$$\begin{aligned} \xi_{k,j} &= \xi_{k-1,0}\xi_{k-1,2j} + \xi_{k-1,2j-1} \\ &= \beta_1^T\sigma_2(\omega_1\xi_{k-1,2j} + \gamma_1\xi_{k-1,0}) + \beta_1^T\sigma_2\left(\omega_1\xi_{k-1,2j-1} + \gamma_1\right), \quad j = 1, 2, ..., 2^{m-k+1}, \end{aligned} \tag{2.39}$$

$$\xi_{k,0} = (\xi_{k-1,0})^2 = \sigma_2(\xi_{k-1,0}). \tag{2.40}$$

Denote $\boldsymbol{\xi}_k = [\xi_{k,1}, \xi_{k,2}, \ldots, \xi_{k,2^{m-k+1}}, \xi_{k,0}]^T$. We have

$$\boldsymbol{\xi}_k = A_{k+1,0}\boldsymbol{x}_k, \quad \boldsymbol{x}_k = \sigma_2(A_{k1}\boldsymbol{\xi}_{k-1} + b_{k1}), \tag{2.41}$$

where $A_{k1}, b_{k1}$ has the same formula as $A_{21}, b_{21}$ given in (2.36) except that the maximum value of $j$ is $2^{m-k+1}$ rather than $2^{m-1}$, and $A_{k+1,0}$ has the same formula as $A_{30}$ given in (2.38) with $\mathbf{1}_{2^{m-1}\times 1}$ replaced by $\mathbf{1}_{2^{m-k+1}\times 1}$. Combining (2.41) and (2.38), we get

$$\boldsymbol{x}_k = \sigma_2(A_k\boldsymbol{x}_{k-1} + b_k), \quad \text{where } A_k = A_{k1}A_{k0}, \quad b_k = b_{k1}. \tag{2.42}$$

4) The output of Layer $m+1$ intermediate variables are:

$$\xi_{m+1,1} = \xi_{m,0}\xi_{m,2} + \xi_{m,1} = \beta_1^T\sigma_2(\omega_1\xi_{m,2} + \gamma_1\xi_{m,0}) + \beta_1^T\sigma_2\left(\omega_1\xi_{m,1} + \gamma_1\right). \tag{2.43}$$

Written into the following form

$$\boldsymbol{\xi}_{m+1} := [\xi_{m+1,1}] = A_{m+2,0}\boldsymbol{x}_{m+1}, \quad \boldsymbol{x}_{m+1} = \sigma_2(A_{m+1,1}\boldsymbol{\xi}_m + b_{m+1,1}), \tag{2.44}$$

we have

$$A_{m+1,1} = \begin{bmatrix} \omega_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \omega_1 & \gamma_1 \end{bmatrix}, \quad b_{m+1,1} = \begin{bmatrix} \gamma_1 \\ \mathbf{0} \end{bmatrix}, \tag{2.45}$$

and

$$A_{m+2,0} = [\beta_1^T \ \beta_1^T], \quad b_{m+2,0} = 0. \tag{2.46}$$

The iteration formula for $\boldsymbol{x}_{m+1}$ is

$$\boldsymbol{x}_{m+1} = \sigma_2(A_{m+1}\boldsymbol{x}_m + b_{m+1}), \quad \text{where } A_{m+1} = A_{m+1,1}A_{m+1,0}, \quad b_{m+1} = b_{m+1,1}. \tag{2.47}$$

5) Since $\boldsymbol{\xi}_{m+1} = f(x)$, the network ends at Layer $m+2$, with $\boldsymbol{x}_{m+2} = \boldsymbol{\xi}_{m+1}$. So we get $A_{m+2} = A_{m+2,0}$, and $b_{m+2} = 0$ from equation (2.44).

For $n < 4$, the procedure can be obtained by removing some sub-steps from the cases $n \geq 4$. From the construction process, we see that the number of layers is $m+2$, the numbers of nodes from Layer 1 to Layer $m+1$ are 6, $8 \times 2^{m-k+1} + 1 (2 \leq k \leq m)$ and 8 respectively, and the number of nonzero weights in $\boldsymbol{A}_j$, $\boldsymbol{b}_j (1 \leq j \leq m+2)$ are not bigger than 10, $(40\times 2^{m-1}+2)+8\times 2^{m-1}$, $(68\times 2^{m-k+1}+1)+4\times 2^{m-k+1} (3 \leq k \leq m)$, 72, 8 respectively. Summing up those number, we reach to the desired conclusion. □

**Remark 2.** *Theorem 1 says we can use a $\sigma_2$ network of scale $\mathcal{O}(\log_2 n)$ to represent $x^n$ exactly. Theorem 2 says that any polynomial of degree less than n can be represented exactly by a $\sigma_2$ neural network with $\lfloor \log_2 n \rfloor + 1$ hidden layers, and no more than $\mathcal{O}(n)$ nonzero weights. Such results are not available for ReLU network and neural networks using other non-polynomial activation functions, such as* logistic, tanh, softplus, sinc *etc. We note that the constants in the two theorems may not be optimal, but the orders of number of layers and number of nonzero weights are sharp.*

### *2.1.2. Error bounds of approximating smooth functions by deep ReQU networks*

Now we analyze the error of approximating general smooth functions using ReQU networks. We first introduce some notations and give a brief review to some classical results of polynomial approximation.

Let $\Omega \subseteq \mathbb{R}^d$ be the domain on which the function to be approximated is defined. For the 1-dimensional case in this section, we focus on $\Omega = I := [-1, 1]$. Similar discussions and results can be extended to $\Omega = [0, \infty]$ and $[-\infty, \infty]$ as well. We denote the set of polynomials with degree up to $N$ defined on $\Omega$ by $P_N(\Omega)$, or simply $P_N$. Let $J_n^{\alpha,\beta}(x)$ be the Jacobi polynomial of degree $n$, $n \in \mathbb{N}_0$, which form a complete set of orthogonal bases in the weighted $L^2_{\omega^{\alpha,\beta}}(I)$ space with respect to weight $\omega^{\alpha,\beta} = (1-x)^\alpha (1+x)^\beta$ for $\alpha, \beta > -1$. To describe functions with high order regularity, we define Jacobi-weighted Sobolev space $B^m_{\alpha,\beta}(I)$ as [43]:

$$B^m_{\alpha,\beta}(I) := \left\{ u : \partial_x^k u \in L^2_{\omega^{\alpha+k,\beta+k}}(I), \quad 0 \leq k \leq m \right\}, \quad m \in \mathbb{N}, \tag{2.48}$$

with norm

$$\|f\|_{B^m_{\alpha,\beta}} := \left( \sum_{k=0}^{m} \|\partial_x^k u\|^p_{L^2_{\omega^{\alpha+k,\beta+k}}} \right)^{1/2}. \tag{2.49}$$

Define the $L^2_{\omega^{\alpha,\beta}}$-orthogonal projection $\pi_N^{\alpha,\beta}$: $L^2_{\omega^{\alpha,\beta}}(I) \to P_N$ as

$$\left( \pi_N^{\alpha,\beta} u - u, v \right)_{\omega^{\alpha,\beta}} = 0, \quad \forall\, v \in P_N. \tag{2.50}$$

A detailed error estimate on the projection error $\pi_N^{\alpha,\beta} u - u$ is given in Theorem 3.35 of [43], by which we have the following theorem on the approximation error of ReQU networks.

**Theorem 3.** *Let $\alpha, \beta > -1$. For any $u \in B^m_{\alpha,\beta}(I)$, there exist a ReQU network $\Phi_N^u$ with $\lfloor \log_2 N \rfloor + 1$ hidden layers, $\mathcal{O}(N)$ nodes, and $\mathcal{O}(N)$ nonzero weights , satisfying the following estimate*

- *if $0 \leq l \leq m \leq N+1$, we have*

$$\left\| \partial_x^l \left( R_{\sigma_2}(\Phi_N^u) - u \right) \right\|_{\omega^{\alpha+l,\beta+l}} \leq c \sqrt{\frac{(N-m+1)!}{(N-l+1)!}} (N+m)^{(l-m)/2} \|\partial_x^m u\|_{\omega^{\alpha+m,\beta+m}}, \tag{2.51}$$

- *if $m > N+1$, we have*

$$\left\| \partial_x^l \left( R_{\sigma_2}(\Phi_N^u) - u \right) \right\|_{\omega^{\alpha+l,\beta+l}} \leq c (2\pi N)^{-1/4} \left( \frac{\sqrt{e/2}}{N} \right)^{N-l+1} \|\partial_x^{N+1} u\|_{\omega^{\alpha+N+1,\beta+N+1}}, \tag{2.52}$$

*where $c \approx 1$ for $N \gg 1$.*

*Proof.* For any given $u \in B^m_{\alpha,\beta}(I)$, there exists a polynomial $f = \pi^{\alpha,\beta}_N u \in P_N$. The projection error $\pi^{\alpha,\beta}_N u - u$ is estimated by Theorem 3.35 in [43], which is exactly (2.51) and (2.52) with $R_{\sigma_2}(\Phi^u_N)$ replaced by $\pi^{\alpha,\beta}_N u$. By Theorem 2, $f$ can be represented exactly by a ReQU network $\Phi^u_N$ with $\lfloor \log_2 N \rfloor + 1$ hidden layers, $\mathcal{O}(N)$ nodes, and $\mathcal{O}(N)$ nonzero weights, i.e. $R_{\sigma_2}(\Phi^u_N) = \pi^{\alpha,\beta}_N u$. We thus obtain estimation (2.51) and (2.52). □

**Remark 3.** *In (2.51) and (2.52), we allow the error measured in high-order derivatives($l \geq 3$), because $R_{\sigma_2}(\Phi^u_N)$ is an exact realization of a polynomial, which is infinitely differentiable. In practice, if $\Phi^u_N$ is a trained network with numerical error, we can not measure the error with derivatives order $\geq 3$, since $\partial^3_x \sigma_2(x)$ is not in $L^2$ space.*

Based on Theorem 3, we can analyze the network complexity of $\varepsilon$-approximation of a given function with certain smoothness. For simplicity, we only consider the case with $l = 0$. The result is given in the following theorem.

**Theorem 4.** *For any given function $f(x) \in B^m_{\alpha,\beta}(I)$ with norm less than 1, where $m$ is either a fixed positive integer or infinity, there exists a ReQU network $\Phi^f_\varepsilon$ with number of layers $L$, number of nonzero weights $N$ satisfying*

- *if $m$ is a fixed positive integer, then $L = \mathcal{O}\left(\frac{1}{m}\log_2 \frac{1}{\varepsilon}\right)$, and $N = \mathcal{O}\big(\varepsilon^{-\frac{1}{m}}\big)$;*
- *if $m = \infty$, i.e. $f$ is analytic, then $L = \mathcal{O}\left(\log_2\left(\log \frac{1}{\varepsilon}\right)\right)$, and $N = \mathcal{O}\big(\frac{1}{\gamma}\log\left(\frac{1}{\varepsilon}\right)\big)$, $\gamma \approx \mathcal{O}\left(\log(\log \frac{1}{\varepsilon})\right)$,*

*can approximate $f$ within an error tolerance $\varepsilon$, i.e.*

$$\|R_{\sigma_2}(\Phi^f_\varepsilon) - f\|_{\omega^{\alpha,\beta}(I)} \leq \varepsilon. \tag{2.53}$$

*Proof.* For a fixed $m$, or $N \gg m$, we obtain from (2.51) that

$$\|R_{\sigma_2}(\Phi^u_N) - u\|_{\omega^{\alpha,\beta}(I)} \leq cN^{-m}\|\partial^m_x u\|_{\omega^{\alpha+m,\beta+m}}. \tag{2.54}$$

By above estimate, we obtain that to achieve an error tolerance $\varepsilon$ to approximate a function with $B^m_{\alpha,\beta}(I)$ norm less than 1, one need to take $N = \left(\frac{c}{\varepsilon}\right)^{\frac{1}{m}}$. For fixed $m$, we have $N = \mathcal{O}\big(\varepsilon^{-\frac{1}{m}}\big)$, the depth of the corresponding ReQU network is $L = \mathcal{O}\left(\frac{1}{m}\log_2 \frac{1}{\varepsilon}\right)$.

For analytic function, by taking $m = \infty$ in equation (2.52), we have

$$\|R_{\sigma_2}(\Phi^u_N) - u\|_{\omega^{\alpha,\beta}(I)} \leq c(2\pi N)^{-\frac{1}{4}}\left(\frac{\sqrt{e/2}}{N}\right)^{N+1}\|u\|_{B^\infty_{\alpha,\beta}} \leq c' e^{-\gamma N}\|u\|_{B^\infty_{\alpha,\beta}}, \tag{2.55}$$

where $c'$ is a general constant, and $\gamma \approx \mathcal{O}(\log N)$ can be larger than any fixed positive number for sufficient large $N$. For simplicity, we can keep it as a constant. To approximate a function with $B^\infty_{\alpha,\beta}(I)$ norm less than 1 with error $\varepsilon = c' e^{-\gamma N}$, one needs to take $N = \frac{1}{\gamma}\log\left(\frac{c'}{\varepsilon}\right)$, which means $N = \mathcal{O}\big(\frac{1}{\gamma}\log\left(\frac{1}{\varepsilon}\right)\big)$. The depth of the corresponding ReQU network is $L = \mathcal{O}\left(\log_2\left(\log \frac{1}{\varepsilon}\right)\right)$.

□

### *2.2. Approximation by deep networks using general rectified power units*

The results of approximation monomials, polynomials and general smooth functions by ReQU networks discussed in subsection 2.1 can be extend to general RePU networks.

To keep the paper short, we only present the results on approximating monomials with RePU in Theorem 5. The other results can be obtained similarly as did in last subsection for ReQU networks.

**Theorem 5.** *Regarding the problem of using $\sigma_s(x)$ $(2 \leq s \in \mathbb{N})$ neural networks to exactly represent monomial $x^n$, $n \in \mathbb{N}$, we have the following results:*

*(1) If $s = n$, the monomial $x^n$ can be realized exactly using a $\sigma_s$ networks having only 1 hidden layer with two nodes.*

*(2) If $1 \le n < s$, the monomial $x^n$ can be realized exactly using a $\sigma_s$ networks having only 1 hidden layer with no more than $2s$ nodes.*

*(3) If $n > s \ge 2$, the monomial $x^n$ can be realized exactly using a $\sigma_s$ networks having $\lfloor \log_s n \rfloor + 2$ hidden layers with no more than $(6s+2)(\lfloor \log_s n \rfloor + 2)$ nodes, no more than $\mathcal{O}(25s^2 \lfloor \log_s n \rfloor)$ nozero weights.*

*Proof.* (1) It is easy to check that $x^s$ has an exact $\sigma_s$ realization given by

$$\rho_s(x) := \sigma_s(x) + (-1)^s \sigma_s(-x) = x^s(x). \tag{2.56}$$

(2) For the case of $1 \le n < s$, we consider the following linear combination

$$a_0 + \sum_{k=1}^{s} a_k \rho_s(x + b_k) = a_0 + \sum_{k=1}^{s} a_k \left( \sum_{j=0}^{s} C_j^s b_k^{s-j} x^j \right) = a_0 + \sum_{j=0}^{s} C_j^s \left( \sum_{k=1}^{s} a_k b_k^{s-j} \right) x^j, \tag{2.57}$$

where $a_0, a_k, b_k, k = 1, \dots, s$ are parameters to be determined. $C_j^s$ are binomial coefficients. Identity the above expression with $x^n$, we obtain the following linear system

$$D_{s+1}\boldsymbol{a} := \begin{bmatrix} 1 & 1 & \cdots & 1 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ b_1^{s-n} & b_2^{s-n} & \cdots & b_s^{s-n} & 0 \\ \vdots & \vdots & & & \vdots \\ b_1^{s-1} & b_2^{s-1} & \cdots & b_s^{s-1} & 0 \\ b_1^{s} & b_2^{s} & \cdots & b_s^{s} & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ . \\ a_s \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ (C_n^s)^{-1} \\ \vdots \\ 0 \end{bmatrix}, \tag{2.58}$$

where the top-left $s \times s$ submatrix of $D_{s+1}$ is a Vandermonde matrix, which is invertible as long as $b_k, k = 1, \dots, s$ are distinct. For simplicity, we choose $b_k, k = 0, \dots, s$ to be equidistant points, then (2.58) is uniquely solvable. Solving for $a_0, \dots, a_s$ we obtain an exact representation of $x^n$ using (2.57), which corresponds to a neural network having one hidden layer with no more than $2s$ $\sigma_s$ units.

For example, for $s = 2$, we may take $b_1 = -1$, $b_1 = 1$, solving equation (2.58) with $n = 1$, we get $a_1 = -\frac{1}{4}$, $a_2 = \frac{1}{4}$, and $a_0 = 0$, thus

$$x = \frac{1}{4}\rho_2(x+1) - \frac{1}{4}\rho_2(x-1).$$

For $s = 3$, if take $b_1 = -1$, $b_2 = 0$, $b_3 = 1$, we obtain

$$\begin{aligned} x &= \frac{1}{6}\big[\rho_3(x-1) - 2\rho_3(x) + \rho_3(x+1)\big] \\ x^2 &= \frac{1}{6}\big[\rho_3(x+1) - \rho_3(x-1)\big] - \frac{1}{3} \end{aligned}$$

(3) Now, we consider the case $n > s \ge 2$, $n \in \mathbb{N}$. For any given quantity $y, z$, using the identity

$$yz = \frac{1}{4}\big[(y+z)^2 - (y-z)^2\big] \tag{2.59}$$

and the fact that $(y+z)^2$, $(y-z)^2$ both can be realized exactly by a one layer $\sigma_s$ network with no more than $2s$ nodes, we conclude that the product $yz$ can be realized by one layer $\sigma_s$ network with no more than $4s$ nodes. To realize $x^n$ by $\sigma_s$, we rewrite $n$ in the following form

$$n = a_m \cdot s^m + a_{m-1} \cdot s^{m-1} + \cdots + a_1 \cdot s + a_0, \tag{2.60}$$

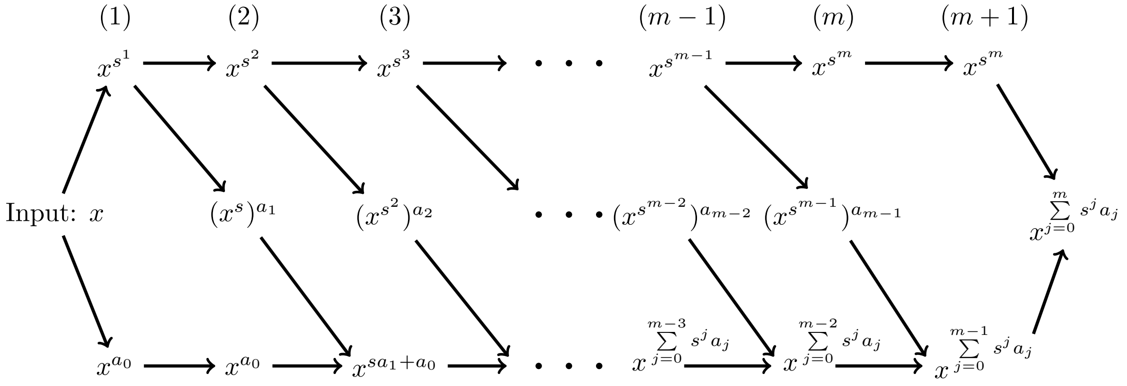
(1)
(2)
(3)
$(m-1)$
$(m)$
$(m+1)$
$x^{s^1}$
$x^{s^2}$
$x^{s^3}$
$x^{s^{m-1}}$
$x^{s^m}$
$x^{s^m}$
Input: $x$
$(x^s)^{a_1}$
$(x^{s^2})^{a_2}$
$(x^{s^{m-2}})^{a_{m-2}}$
$(x^{s^{m-1}})^{a_{m-1}}$
$x^{\sum_{j=0}^{m} s^j a_j}$
$x^{a_0}$
$x^{a_0}$
$x^{sa_1+a_0}$
$x^{\sum_{j=0}^{m-3} s^j a_j}$
$x^{\sum_{j=0}^{m-2} s^j a_j}$
$x^{\sum_{j=0}^{m-1} s^j a_j}$

Figure 2: A schematic diagram for $\sigma_s$ network realization of $x^n$, $n > s$. $(j)$ represents the $j$-th hidden layer of intermediate variables.

where $a_j \in \{0, 1, \ldots, s-1\}$ for $j = 0, 1, ..., m-1$ and $a_m = 1$. So we have

$$x^n = (x^{s^m})^{a_m}(x^{s^{m-1}})^{a_{m-1}} \cdots (x^s)^{a_1}(x)^{a_0} \tag{2.61}$$

Define $\xi_k = x^{s^k}$, $z_{k+1} = (\xi_k)^{a_k}$, $k = 0, 1, \ldots, m$, and

$$y_2 = x^{a_0}, \qquad y_{k+2} = z^{k+1} y_{k+1} \ \big( = (x^{s^k})^{a_k} y_{k+1} \big), \quad \text{for } k = 1, \ldots, m, \tag{2.62}$$

we have $y_{m+2} = x^n$. The equation (2.62) can be regarded as an iteration scheme, with iteration variables $\xi_k, y_k, z_k$, where subscript $k$ stands for iteration step. A schematic diagram for this iteration is given in Figure 2. Different to Theorem 1, for $s > 2$, we need a deep $\sigma_s$ neural network with $m+2$ hidden layers to realize $x^n, n > s$, due to the introduction of intermediate variables $z_k$. In each layer, we need no more than $2 + 2s + 4s$ activation nodes to calculate $\xi_{k+1} = \rho_s(\xi_k)$, $z_{k+1} = (\xi_k)^{a_k}$, and $y_{k+1} = z_k y_k$. So in total we need no more than $(6s+2)(m+2) = \mathcal{O}(6s \log_s n)$ nodes. A direct calculation shows that the number of nonzero weights in the network is no more than $\mathcal{O}(25s^2 \log_s n)$. The theorem is proved. □

## 3. Approximation of multivariate smooth functions

In this section, we discuss how to approximate multivariate smooth functions by ReQU networks. Similar to the univariate case, we first study the representation of polynomials then discuss the approximation error of general smooth functions.

### *3.1. Deep ReQU network representations of multivariate polynomials*

**Theorem 6.** *If $f(x)$ is a multivariate polynomial with* total *degree $n$ on $\mathbb{R}^d$, then there exists a $\sigma_2$ neural network having $d\lfloor \log_2 n \rfloor + d$ hidden layers with no more than $\mathcal{O}(C_d^{n+d})$ activation functions and nonzero weights, can represent $f$ with no error.*

*Proof.* 1) We first consider the 2-dimensional case. Suppose $f(x,y) = \sum_{i+j=0}^{n} a_{ij} x^i y^j$, and $n \geq 4$ (the results for $n \leq 3$ are similar but easier, so skipped here). To represent $f(x,y)$ exactly with a $\sigma_2$ neural network based the results on 1-dimensional case given in Theorem 2, we first rewrite $f(x,y)$ as

$$f(x,y) = \sum_{i=0}^{n} \left( \sum_{j=0}^{n-i} a_{ij} y^j \right) x^i =: \sum_{i=0}^{n} a_i^y x^i, \quad \text{where} \quad a_i^y = \sum_{j=0}^{n-i} a_{ij} y^j. \tag{3.1}$$

So to realize $f(x, y)$, we can first realize $a_i^y$, $i = 0, \dots, n-1$ using $n$ small $\sigma_2$ networks $\Phi_i$, $i = 0, \dots, n-1$, i.e. $R_{\sigma_2}(\Phi_i)(y) = a_i^y$ for given input $y$; then use a $\sigma_2$ network $\Phi_n$ to realize the 1-dimensional polynomials $f(x, y) = \sum_{i=0}^n a_i^y x^i$. There are two places need some technique treatments, the details are given below.

(1) The network $\Phi_n$ takes $a_i^y, i = 0, \dots, n$ and $x$ as input. So these quantities must be presented at the same layer of the overall neural network, because we do not want connections over disjointed layers. By Theorem 2, the largest depth of networks $\Phi_i, i = 0, \dots, n-1$ is $\lfloor \log_s n \rfloor + 2$, so we can lift $x$ to layer $\lfloor \log_s n \rfloor + 2$ using multiple $id_X(\cdot)$ operations. Similarly, we also keep a record of input $y$ in each layer using multiple $id_X(\cdot)$ operations, such that $\Phi_i, i = 1, \dots, n-1$ can start from appropriate layer and generate output exactly at layer $\lfloor \log_s n \rfloor + 2$. The overall cost for recording $x, y$ in layers $1, \dots, \lfloor \log_s n \rfloor + 2$ is $\mathcal{O}(\lfloor \log_s n \rfloor + 2)$, which is small comparing to the number of coefficients $C_d^{n+d}$.

(2) While realizing $\sum_{i=0}^n a_i^y x^i$, the coefficients $a_i^y, i = 0, \dots n$ are network input instead of fixed parameters. So when applying the network construction given in Theorem 2, we need to modify the structure of the first layer of the network. More precisely, equation (2.29) in Theorem 2 should be changed to

$$\xi_{1,j}^y = a_{2j-1}^y x + a_{2j-2}^y = \beta_1^T \sigma_2 \left( \omega_1 x + \gamma_1 a_{2j-1} \right) + \beta_1^T \sigma_2 \left( \omega_1 a_{2j-2} + \gamma_1 \right), \quad j = 1, 2, ..., 2^m. \tag{3.2}$$

So the number of nodes for the first layer changed from 6 to $4 + 8 \times 2^m$, the number of nonzero weights for the first layer changed from 10 to $16 \times 2^m + 4$. So the number of hidden layers, number of nodes and nonzero weights of $\Phi_n$ can be bounded by $\lfloor \log_s n \rfloor + 1$, $17n$, and $77n$.

Assembling $\Phi_0, \dots, \Phi_n$, the overall network to represent $f(x, y)$ has $2\lfloor \log_s n \rfloor + 3$ layers with number of nodes no more than

$$\sum_{j=0}^{n-1} 9(n-j) + 17n + 8(m+2) = 9\frac{n(n+1)}{2} + 17n + 8m + 16 = \mathcal{O}(C_d^{n+d}),$$

and number of weights no more than

$$\sum_{j=0}^{n-1} 61(n-j) + 77n + 16(m+2) \times 2 + 12n = 61\frac{n(n+1)}{2} + 89n + 32m + 64 = \mathcal{O}(C_d^{n+d}).$$

Thus, we proved that the theorem is true for the case $d = 2$.

2) The case $d > 2$ can be proved by mathematical induction using the similar procedure as done for $d = 2$ case.

□

Using a similar approach as in Theorem 6, one can easily prove the following theorem.

**Theorem 7.** *For a polynomials $f_N$ in a tensor product space $Q_N^d(I_1 \times \cdots \times I_d) := P_N(I_1) \otimes \cdots \otimes P_N(I_d)$, there exists a $\sigma_2$ network having $d\lfloor \log_2 N \rfloor + d$ hidden layers with no more than $\mathcal{O}(N^d)$ activation functions and nonzero weights, can represent $f_N$ with no error.*

### *3.2. Error bounds of approximating multivariate smooth functions by deep ReQU networks*

Now we analyze the error of approximating general multivariate smooth functions using ReQU networks.

For a vector $\boldsymbol{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$, we define $|\boldsymbol{x}|_1 := |x_1| + \ldots + |x_d|$, $|\boldsymbol{x}|_\infty := \max_{i=1}^d |x_i|$. Define high dimensional Jacobi weight $\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}} := \omega^{\alpha_1,\beta_1} \cdots \omega^{\alpha_d,\beta_d}$. We define multidimensional Jacobi-weighted Sobolev space $B_{\alpha,\beta}^m(I^d)$ as [43]:

$$B_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m(I^d) := \left\{ u(\boldsymbol{x}) \mid \partial_{\boldsymbol{x}}^{\boldsymbol{k}} u := \partial_{x_1}^{k_1} \cdots \partial_{x_d}^{k_d} u \in L^2_{\omega^{\boldsymbol{\alpha}+\boldsymbol{k},\boldsymbol{\beta}+\boldsymbol{k}}}(I^d), \quad \boldsymbol{k} \in \mathbb{N}_0^d, \ |\boldsymbol{k}|_1 \le m \right\}, \quad m \in \mathbb{N}_0, \tag{3.3}$$

with norm and semi-norm

$$\|u\|_{B_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m} := \left( \sum_{0 \le \ |\boldsymbol{k}|_1 \le m} \left\| \partial_{\boldsymbol{x}}^{\boldsymbol{k}} u \right\|^2_{L^2_{\omega^{\boldsymbol{\alpha}+\boldsymbol{k},\boldsymbol{\beta}+\boldsymbol{k}}}} \right)^{1/2}, \quad |u|_{B_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m} := \left( \sum_{|\boldsymbol{k}|_1 = m} \left\| \partial_{\boldsymbol{x}}^{\boldsymbol{k}} u \right\|^2_{L^2_{\omega^{\boldsymbol{\alpha}+\boldsymbol{k},\boldsymbol{\beta}+\boldsymbol{k}}}} \right)^{1/2}. \tag{3.4}$$

Define the $L^2_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}}$-orthogonal projection $\pi_N^{\boldsymbol{\alpha},\boldsymbol{\beta}}$: $L^2_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}}(I^d) \to Q_N^d(I^d)$ as

$$\left(\pi_N^{\boldsymbol{\alpha},\boldsymbol{\beta}} u - u, v\right)_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}} = 0, \quad \forall\, v \in P_N^d(I^d). \tag{3.5}$$

Then for $u \in B^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$, we have the following error estimate [43]:

$$\|\pi_N^{\boldsymbol{\alpha},\boldsymbol{\beta}} u - u\|_{L^2_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}}(I^d)} \le cN^{-m}|u|_{B^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}}, \quad 1 \le m \le N, \tag{3.6}$$

where $c$ is a general constant. Combining (3.6) and Theorem 7, we reach to the following upper bound for the $\varepsilon$-approximation of functions in $B^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$ space.

**Theorem 8.** *For any $u \in B^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$, with $|u|_{B^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)} \le 1$, there exists a $\sigma_2$ neural network $\Phi^u_\varepsilon$ having $\mathcal{O}\left(\frac{d}{m}\log_2\frac{1}{\varepsilon} + d\right)$ layers with no more than $\mathcal{O}\left(\varepsilon^{-d/m}\right)$ nodes and nonzero weights, approximate $u$ with $L^2_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}}(I^d)$-error less than $\varepsilon$, i.e.*

$$\|R_{\sigma_2}(\Phi^u_\varepsilon) - u\|_{L^2_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}}(I^d)} \le \varepsilon. \tag{3.7}$$

Results similar to Theorem 8 can be obtained for the approximation on $\mathbb{R}^d$ and $(\mathbb{R}^+)^d$ using the Hermite and Laguerre polynomial projection.

**Remark 4.** *Comparing Theorem 8 with Theorem 1 in [18], we see that the number of computational units and nonzero weights needed by a ReQU network to approximate a function $u \in B^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$ with an error tolerance $\varepsilon$ is less than that needed by a ReLU network. The ReLU network is $\log\frac{1}{\varepsilon}$ times larger than corresponding ReQU network. For low accuracy approximation, the factor $\mathcal{O}(\log\frac{1}{\varepsilon})$ is not very big, but for high accuracy approximations, this factor can as large as several dozens, which is expected to make significant difference in large scale computations.*

## 4. High-dimensional smooth functions with sparse polynomial approximations

In last section, we showed that for a $d$-dimensional functions with partial derivatives up to order $m$ in $L^2(I^d)$ can be approximated within error $\varepsilon$ by a ReQU neural network with complexity $\mathcal{O}(\varepsilon^{-d/m})$. When $m$ is fixed or much smaller than $d$, the network complexity has an exponential dependence on $d$. However, in a lot of applications, high-dimensional problem may have low intrinsic dimension (see e.g. [44][45]). One particular example is high-dimensional tensor product functions(or linear combinations of finite terms of tensor product functions), which can be well approximated by a *hyperbolic cross* or *sparse grid* truncated series.

### *4.1. A brief review on hyperbolic cross approximations and sparse grids*

Sparse grids were originally introduced by S. A. Smolyak[21] to integrate or interpolate high dimensional functions. Hyperbolic cross approximation is a technique similar to sparse grids but without the concept of grids. We introduce hyperbolic cross approximation by considering a tensor product function: $f(\boldsymbol{x}) = f_1(x_1)f_1(x_2)\cdots f_d(x_d)$. Suppose that $f_1, \ldots, f_d$ have similar regularity that can be well approximated by using a set of orthonormal bases $\{\phi_k, k = 1, 2, \ldots.\}$ as

$$f_i(x) = \sum_{k=0}^{\infty} b_k^{(i)}\phi_k(x), \quad |b_k^{(i)}| \le c\bar{k}^{-r}, \quad i = 1, \ldots, d, \tag{4.1}$$

where $c$ is a general constant, $r \ge 1$ is a constant depending on the regularity of $f_i$, $\bar{k} := \max\{1, k\}$. So we have an expansion for $f$ as

$$f(\boldsymbol{x}) = \prod_{i=1}^{d}\left(\sum_{k=0}^{\infty} b_k^{(i)}\phi_k(x_i)\right) = \sum_{\boldsymbol{k}\in\mathbb{N}_0^d} b_{\boldsymbol{k}}\phi_{\boldsymbol{k}}(\boldsymbol{x}), \quad \text{where } |b_{\boldsymbol{k}}| = \left|b_{k_1}^{(1)}\cdots b_{k_d}^{(d)}\right| \le c^d(\bar{k}_1\cdots\bar{k}_d)^{-r}. \tag{4.2}$$

Thus, to have a best approximation of $f(\boldsymbol{x})$ using finite terms, one should take

$$f_N := \sum_{\boldsymbol{k}\in\chi_N^d} b_{\boldsymbol{k}}\phi_{\boldsymbol{k}}(\boldsymbol{x}), \tag{4.3}$$

where

$$\chi_N^d := \left\{\boldsymbol{k} = (k_1, \dots, k_d) \in \mathbb{N}_0^d \mid \bar{k}_1 \cdots \bar{k}_d \le N\right\} \tag{4.4}$$

is the hyperbolic cross index set. We call $f_N$ defined by (4.3) a hyperbolic cross approximation of $f$.

For general functions defined on $I^d$, we choose $\phi_{\boldsymbol{k}}$ to be multivariate Jacobi polynomials $J_{\boldsymbol{n}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}$, and define the hyperbolic cross polynomial space as

$$X_N^d := \mathrm{span}\{\, J_{\boldsymbol{n}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}, \quad \boldsymbol{n} \in \chi_N^d \,\}. \tag{4.5}$$

Note that the definition of $X_N^d$ doesn't depends on $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. $\{J_{\boldsymbol{n}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}\}$ is used to served as a set of bases for $X_N^d$. To study the error of hyperbolic cross approximation, we define Jacobi-weighted Korobov-type space

$$\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m(I^d) := \left\{\, u(\boldsymbol{x}) \;:\; \partial_{\boldsymbol{x}}^{\boldsymbol{k}} u \in L^2_{\omega^{\boldsymbol{\alpha}+\boldsymbol{k},\boldsymbol{\beta}+\boldsymbol{k}}}(I^d),\ 0 \le\ |\boldsymbol{k}|_\infty \le m \,\right\}, \quad \text{for } m \in \mathbb{N}_0, \tag{4.6}$$

with norm and semi-norm

$$\|u\|_{\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m} := \left(\sum_{0\le\ |\boldsymbol{k}|_\infty\le m} \left\|\partial_{\boldsymbol{x}}^{\boldsymbol{k}} u\right\|^2_{L^2_{\omega^{\boldsymbol{\alpha}+\boldsymbol{k},\boldsymbol{\beta}+\boldsymbol{k}}}}\right)^{1/2}, \quad |u|_{\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m} := \left(\sum_{|\boldsymbol{k}|_\infty=m} \left\|\partial_{\boldsymbol{x}}^{\boldsymbol{k}} u\right\|^2_{L^2_{\omega^{\boldsymbol{\alpha}+\boldsymbol{k},\boldsymbol{\beta}+\boldsymbol{k}}}}\right)^{1/2}. \tag{4.7}$$

For any given $u \in \mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^0 (= B_{\boldsymbol{\alpha},\boldsymbol{\beta}}^0)$, the hyperbolic cross approximation can be defined as a projection as

$$(\pi_{N,H}^{\boldsymbol{\alpha},\boldsymbol{\beta}} u - u, v)_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}} = 0, \quad \forall\, v \in X_N^d. \tag{4.8}$$

Then we have the following error estimate about the hyperbolic cross approximation [24]:

$$\|\partial_{\boldsymbol{x}}^{\boldsymbol{l}}(\pi_{N,H}^{\boldsymbol{\alpha},\boldsymbol{\beta}} u - u)\|_{\omega^{\boldsymbol{\alpha}+\boldsymbol{l},\boldsymbol{\beta}+\boldsymbol{l}}} \le D_1 N^{|\boldsymbol{l}|_\infty - m} |u|_{\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m}, \quad 0 \le \boldsymbol{l} \le \boldsymbol{m},\ \boldsymbol{m} \ge 1, \tag{4.9}$$

where $D_1$ is a constant independent of $N$. It is known that the cardinality of $\chi_N^d$ is of order $\mathcal{O}(N(\log N)^{d-1})$. The above error estimate says that to approximate a function $u \in \mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m$ with an error tolerance $\varepsilon$, one need no more than $\mathcal{O}\left(\varepsilon^{-1/m}(\frac{1}{m}\log\frac{1}{\varepsilon})^{d-1}\right)$ Jacobi polynomials, the exponential dependence on $d$ is weakened (cp. Theorem 8). To remove the exponential term $(\log\frac{1}{\varepsilon})^{d-1}$, one may consider a more general sparse polynomial space[24]:

$$X_{N,\gamma}^d := \mathrm{span}\left\{\, J_{\boldsymbol{n}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}, \quad (\Pi_{i=1}^d \bar{n}_i)|\boldsymbol{n}|_\infty^{-\gamma} \le N^{1-\gamma} \,\right\}, \quad -\infty \le \gamma < 1. \tag{4.10}$$

In particular, $X_{N,0}^d = X_N^d$ is the hyperbolic cross space defined in (4.5), and $X_{N,-\infty}^d := \mathrm{span}\left\{\, J_{\boldsymbol{n}}^{\boldsymbol{\alpha},\boldsymbol{\beta}},\ |\boldsymbol{n}|_\infty \le N \,\right\}$ is the standard full grid. For $0 < \gamma < 1$, it is known that [23]:

$$\mathrm{Card}(X_{N,\gamma}^d) = C(\gamma, d)N, \quad 0 < \gamma < 1, \tag{4.11}$$

where $C(\gamma, d)$ is a constant that depends on $\gamma$ and $d$ but is independent of $N$. We call $X_{N,\gamma}^d, 0 < \gamma < 1$ optimized hyperbolic cross polynomial space. It is proved by Shen and Wang [24] that the $L^2_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}}$-orthogonal projection $\pi_{N,\gamma}^{\boldsymbol{\alpha},\boldsymbol{\beta}}$ from Korobov space to $X_{N,\gamma}^d$ satisfies the following estimate (see Theorem 2.3 in [24]):

$$\|\pi_{N,\gamma}^{\boldsymbol{\alpha},\boldsymbol{\beta}} u - u\|_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}} \le D_2 N^{-m(1-\gamma(1-\frac{1}{d}))} |u|_{\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m}, \quad 0 < \gamma < 1, \tag{4.12}$$

where $D_2$ is a constant independent of $N$. From (4.11) and (4.12), we get that to approximate a function $u \in \mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m$ with an error tolerance $\varepsilon$, one need no more than $\mathcal{O}\left(\varepsilon^{-1/m(1-\gamma(1-\frac{1}{d}))}\right)$ Jacobi polynomials. We will later use this estimate to give a better upper bound of approximating functions in $\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m$ using deep ReQU networks.

In practice, the exact hyperbolic cross projection is not easy to calculate. An alternate approach is the sparse grids, which use hierarchical interpolation schemes to build an hyperbolic cross like approximation of high dimensional functions. To define sparse grids for $I^d$, we first define the underlying 1-dimensional interpolations. Given a series of interpolation point set $\mathcal{X}^i = \{x_1^i, \cdots, x_{m_i}^i\} \subseteq [-1,1]$, $m_i = \mathrm{Card}(\mathcal{X}^i)$, $i = 1, 2, \ldots$, with $0 < m_1 < m_2 < \cdots$, the interpolation on $\mathcal{X}^i$ for $f \in C^0(I)$ is defined as

$$\mathcal{U}^i(f) = \sum_{j=1}^{m_i} f(x_j^i)\ell_j^i(x), \tag{4.13}$$

where $\ell_j^i(x) \in P_{m_i-1}([-1,1])$ are the Lagrange interpolation bases. The sparse grid interpolation for high-dimension function $f \in C^0(I^d)$ is defined as [21]:

$$\mathcal{A}(q,d)(f) = \sum_{d=|\boldsymbol{i}|_1 \le q} \Big(\Delta^{i_1} \otimes \cdots \otimes \Delta^{i_d}\Big)(f), \quad q \ge d, \tag{4.14}$$

where $\Delta^i = \mathcal{U}^i - \mathcal{U}^{i-1}$, $\boldsymbol{i} \in \mathbb{N}^d$. For convenience, we define $\mathcal{U}^0 := 0$, $m_0 = 0$, $\mathcal{X}^0 = \emptyset$. Formally, (4.14) can be defined on any grids $\{\, \mathcal{X}^i,\, i = 1, 2, \ldots, q-d+1 \,\}$. However, to have a one-to-one transform between the values on interpolation points and the coefficients of linear independent bases in the interpolation space, we need $\{\, \mathcal{X}^i,\, i = 1, 2, \ldots, q-d+1 \,\}$ to be nested, i.e. $\mathcal{X}^1 \subset \mathcal{X}^2 \subset \cdots \mathcal{X}^{q-d+1}$. Fast transforms between physical values and interpolation coefficients always exist for sparse grid interpolations using nested grids [29]. Define sparse grid index set as

$$\mathcal{I}_d^q := \bigcup_{d \le |\boldsymbol{i}|_1 \le q} \tilde{\mathcal{I}}^{i_1} \times \cdots \times \tilde{\mathcal{I}}^{i_d}, \quad \text{where } \tilde{\mathcal{I}}^k := \mathcal{I}^k \setminus \mathcal{I}^{k-1}, \quad \mathcal{I}^k = \{\, 1, 2, \ldots, m_i \,\}. \tag{4.15}$$

Then the set of the sparse grid interpolation points and the corresponding interpolation space are given as

$$\mathcal{X}_d^q = \bigcup_{d=|\boldsymbol{i}|_1 \le q} \Big((\mathcal{X}^{i_1} \setminus \mathcal{X}^{i_1-1}) \otimes \cdots \otimes (\mathcal{X}^{i_1} \setminus \mathcal{X}^{i_1-1})\Big), \quad q \ge d, \tag{4.16}$$

$$V_d^q = \mathrm{span}\{\, \tilde{\phi}_{\boldsymbol{k}}(\boldsymbol{x}),\, \boldsymbol{k} \in \mathcal{I}_d^q \,\} \quad q \ge d, \tag{4.17}$$

where $\tilde{\phi}_{\boldsymbol{k}}$ can be chosen as the hierarchical interpolation bases defined in [29], or the Lagrange-type $d$-dimensional interpolation polynomial on points $\mathcal{X}_d^d$, which takes value 1 on $\boldsymbol{k}$-th interpolation point and 0 on other points.

A commonly used 1-dimensional scheme is the Chebyshev-Gauss-Lobatto scheme, which uses the extrema of the Chebyshev polynomials as interpolation points:

$$x_j^i = -\cos\left(\frac{(j-1)\pi}{m_i - 1}\right), \quad j = 1, 2, \cdots, m_i. \tag{4.18}$$

In order to obtain nested sets of points, $m_i$ are chosen as

$$m_i = \begin{cases} 1, & i = 1, \\ 2^{i-1} + 1, & i > 1, \end{cases} \tag{4.19}$$

with $x_1^1 := 0$. Define

$$F_d^k := \{\, f : [-1,1]^d \to \mathbb{R} \mid D^{\boldsymbol{\alpha}} f \in C([-1,1]^d),\ \forall\, |\boldsymbol{\alpha}|_\infty \le k \,\}. \tag{4.20}$$

Then for any function $f \in F_d^k$, with $\|f\|_{F_d^k} \le 1$, the interpolation error on the above Chebyshev sparse grids are bounded as [26]:

$$\|f - \mathcal{A}(q,d)f\|_{L^\infty} \le c_{d,k} 2^{-kq} q^{2d-1} \le c_{d,k} n^{-k} (\log n)^{(k+2)(d-1)+1}, \tag{4.21}$$

where $n = \mathrm{Card}(\mathcal{X}_d^q) = \mathrm{Card}(\mathcal{I}_d^q) = \mathcal{O}(2^q q^{d-1})$ is the number of points in the sparse grids, and $c_{d,k}$ is a constant depends on $d, k$ only. Note that if other norm instead of the $L^\infty$ norm is used, one can improve the result a little bit, but no results with error bound smaller than $\mathcal{O}(n^{-k})$ is known.

### *4.2. Error bounds of deep ReQU network approximation for multivariate functions with sparse structures*

Now we discuss the ReQU network approximation of high-dimensional smooth functions with sparse polynomial expansions, which takes hyperbolic cross and sparse grid polynomial expansions as examples. We introduce a concept of *complete* polynomial space first. A linear polynomial space $P_C$ is said to be complete if it satisfies the following: if $p(\boldsymbol{x}) \in P_C$, then $\partial_{\boldsymbol{x}}^{\boldsymbol{k}} p(\boldsymbol{x}) \in P_C$ for any $\boldsymbol{k} \in \mathbb{N}_0^d$, where $p(\boldsymbol{x})$ is a $d$-dimensional a polynomial. It is easy to verify that the hyperbolic cross polynomial space $X_N^d$, the sparse grid polynomial interpolation space $V_d^q$, and the optimized hyperbolic cross space $X_{N,\gamma}^d$ are all complete. For a complete polynomial space, we have the following ReQU network representation results.

**Theorem 9.** *Let $P_C$ be a complete linear space of $d$-dimensional polynomials with dimension $n$, then for any function $f \in P_C$, there exists a $\sigma_2$ neural network having no more than $\sum_{i=1}^d \lfloor \log_2 N_i \rfloor + d$ hidden layers, no more than $\mathcal{O}(n)$ activation functions and nonzero weights, can represent $f$ exactly. Here $N_i$ is the maximum polynomial degree in $i$th direction in $P_C$.*

*Proof.* The proof is similar to Theorem 6. First, $f$ can be written as linear combinations of monomials.

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \chi_C} a_{\boldsymbol{k}} \boldsymbol{x}^{\boldsymbol{k}}, \tag{4.22}$$

where $\chi_C$ is the index set of $P_C$ with cardinality $n$. Then we rearrange the summation as

$$f(\boldsymbol{x}) = \sum_{k_d=0}^{N_d} a_{k_d}^{x_1 x_2 \cdots x_{k_{d-1}}} x_d^{k_d}, \qquad a_{k_d}^{x_1 x_2 \cdots x_{k_{d-1}}} := \sum_{(k_1,k_2,\ldots,k_{d-1}) \in \chi_C^{k_d}} a_{k_1 k_2 \cdots k_{d-1}} x_1^{k_1} x_2^{k_2} \cdots x_{d-1}^{k_{d-1}}, \tag{4.23}$$

where $\chi_C^{k_d}$ are $d-1$ dimensional complete index sets that depend on the index $k_d$. If each $a_{k_d}^{x_1 x_2 \cdots x_{k_{d-1}}}$, $k_d = 0, 1, \ldots, N_d$ can be exactly represented by a $\sigma_2$ network with no more than $\sum_{i=1}^{d-1} \lfloor \log_2 N_i \rfloor + (d-1)$ hidden layers, no more $\mathcal{O}(\mathrm{Card}(\chi_C^{k_d}))$ nodes and nonzero weights, then $f(x)$ can be exactly represented by a $\sigma_2$ neural network with no more $\sum_{i=1}^d \lfloor \log_2 N_i \rfloor + d$ hidden layers, no more than $\mathcal{O}(n)$ nodes and nonzero weights, since the operation $\sum_{k_d=0}^{N_d} a_{k_d}^{x_1 x_2 \cdots x_{k_{d-1}}} x_d^{k_d}$ can be realized exactly by a $\sigma_2$ network with $\lfloor \log_2 N_d \rfloor + 1$ hidden layers and no more than $\mathcal{O}(N_d)$ nodes and nonzeros operations. So, by mathematical induction, we only need to prove that when $d = 1$ the theorem is satisfied, which is true by Theorem 2. □

**Remark 5.** *According to Theorem 9, we have that:*

1) *For any $f \in X_N^d$, there exists a ReQU network having no more than $d\lfloor \log_2 N \rfloor + d$ hidden layers, no more than $\mathcal{O}(N(\log N)^{d-1})$ activation functions and nonzero weights, can represent $f$ with no error.*

2) *For any $f \in X_{N,\gamma}^d$ with $0 < \gamma < 1$, there exists a ReQU network having no more than $d\lfloor \log_2 N \rfloor + d$ hidden layers, no more than $\mathcal{O}(N)$ activation functions and nonzero weights, can represent $f$ with no error.*

3) *For any $f \in V_d^q$, there exists a ReQU network having no more than $d(q-d+2)$ hidden layers, no more than $\mathcal{O}(2^q q^{d-1})$ activation functions and nonzero weights, can represent $f$ with no error.*

Combine the results in Remarks 5 with (4.9),(4.12) and (4.21), we obtain the following theorem.

**Theorem 10.** *We have following results for ReQU network approximation of functions in $\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m(I^d)$, $m \geq 1$ and $F_d^k(I^d)$, $k \geq 1$:*

1) *For any function $u \in \mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m(I^d)$, $m \geq 1$ with $|u|_{\mathcal{K}_{\boldsymbol{\alpha},\boldsymbol{\beta}}^m} \leq 1/D_1$, any $\varepsilon \geq 0$, there exists a ReQU network $\Phi_\varepsilon^u$ with no more than $\frac{d}{m} \log_2 \frac{1}{\varepsilon} + d$ hidden layers, no more than $\mathcal{O}\big(\varepsilon^{-1/m}(\frac{1}{m} \log \frac{1}{\varepsilon})^{d-1}\big)$ nodes and nonzero weights, such that*
$$\|R_{\sigma_2}(\Phi_\varepsilon^u) - u\|_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}} \leq \varepsilon. \tag{4.24}$$

*2) For any function $u \in \mathcal{K}^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$, $m \geq 1$ with $|u|_{\mathcal{K}^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}} \leq 1/D_2$, any $\varepsilon \geq 0$, there exists a ReQU network $\Phi^u_\varepsilon$ with no more than $\frac{d}{m(1-\gamma(1-\frac{1}{d}))}\log_2\frac{1}{\varepsilon} + d$ hidden layers, no more than $\mathcal{O}\big(\varepsilon^{-1/m(1-\gamma(1-\frac{1}{d}))}\big)$ nodes and nonzero weights, such that*

$$\|R_{\sigma_2}(\Phi^u_\varepsilon) - u\|_{\omega^{\boldsymbol{\alpha},\boldsymbol{\beta}}} \leq \varepsilon. \tag{4.25}$$

*3) For any function $f \in F^k_d(I^d)$, $k \geq 1$ with $\|f\|_{F^k_d} \leq 1$, any $\varepsilon \geq 0$, there exists a ReQU network $\Psi^f_\varepsilon$ with no more than $\mathcal{O}\left(d\frac{1+\delta}{k}\log_2\frac{1}{\varepsilon} + d\right)$ hidden layers, no more than $\mathcal{O}\big(\varepsilon^{-(1+\delta)/k}(\frac{1+\delta}{k}\log_2\frac{1}{\varepsilon})^{d-1}\big)$ nodes and nonzero weights, such that*

$$\|R_{\sigma_2}(\Psi^f_\varepsilon) - f\|_{L^\infty} \leq \varepsilon, \tag{4.26}$$

*where $\delta > 0$ can be taken very close to $0$ for small enough $\varepsilon$.*

**Remark 6.** *Taking $m = 2$ in Theorem 10, we obtain the following result: For any function $u \in \mathcal{K}^2_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$, with $|u|_{\mathcal{K}^2_{\boldsymbol{\alpha},\boldsymbol{\beta}}} \leq 1/D_1$, there exists a ReQU network $\Phi^u_\varepsilon$ with no more than $\frac{d}{2}\log_2\frac{1}{\varepsilon} + d$ hidden layers, no more than $\mathcal{O}\big(\varepsilon^{-1/2}(\frac{1}{2}\log\frac{1}{\varepsilon})^{d-1}\big)$ nodes and nonzero weights, approximate $u$ with a tolerance $\varepsilon$. The result of using ReLU networks approximating similar functions is recently given by Montanelli and Du [39]. Their conclusion is: for a ReLU network to approximate a function in $\mathcal{K}^2_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$ with tolerance $\varepsilon$, the number of layers required is $\mathcal{O}(|\log_2\varepsilon|\log_2 d)$, the number of nonzero weights required is $\mathcal{O}(\varepsilon^{-\frac{1}{2}}|\log_2\varepsilon|^{\frac{3}{2}(d-1)+1}\log_2 d)$. Comparing the two results, we find that, while the number of layers required by ReQU networks might be larger than ReLU networks, the overall complexity of the ReQU network is $|\log_2\varepsilon|^d$ times smaller than the ReLU network.*

**Remark 7.** *When one use optimized hyperbolic cross polynomial approximation for funcion in $\mathcal{K}^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}(I^d)$, with $|u|_{\mathcal{K}^m_{\boldsymbol{\alpha},\boldsymbol{\beta}}} \leq 1/D_2$, the exponential growth on $d$ with a base related to $1/\varepsilon$ in the required ReQU network size is removed. Thus, in this case the curse of dimensionality is overcome. We note that, the constant $D_2$ and the implicit constant hidden in the big $\mathcal{O}$ notation, still depend on $d$, but independent of $\varepsilon$.*

## 5. Conclusion and future work

In this paper, we give constructive proofs to error bounds of approximating smooth functions by deep neural networks using RePU as activation functions. The proofs rely on the fact that polynomials can be represented by RePU networks with no approximation error. We construct several optimal algorithms for such representations, in which polynomials of degree no more than $n$ are converted into a ReQU network with $\mathcal{O}(\log_2 n)$ layers, and the size of the network is of the same scale as the polynomial space to be approximated. Then by using the classical polynomial approximation theory, we obtain error bounds for ReQU networks approximating smooth functions, which show clear advantages of using ReQU activation function, comparing to the existing results for ReLU networks. In general, the ReLU network required to approximate a functions with finite-order continuous, is $\mathcal{O}(\log\frac{1}{\varepsilon})$ times larger the the corresponding ReQU network. Here $\varepsilon$ is the approximation error. To achieve $\varepsilon$-approximation for analytic functions, the number of layer of ReQU network required is $\mathcal{O}(\log_2\log\frac{1}{\varepsilon})$, while the corresponding number is $\mathcal{O}(\log\frac{1}{\varepsilon})$ for ReLU network. For high dimensional functions with bounded mixed derivatives, we give error bounds that has a weaker exponentially dependence on $d$, by using hyperbolic cross/sparse grid spectral approximation, in particular if optimized hyperbolic cross polynomial projections are used, the curse of dimensionality is overcome. The complexity of ReQU networks that required to achieve $\varepsilon$-approximation to functions with bounded mixed derivatives up to 2, is much smaller than the corresponding ReLU networks as well. These results hold for deep networks with non-rectified power units. The use of rectified units gives the neural network the ability to approximate piecewise smooth functions efficiently.

The advantage of using *deep* over *shallow* neural ReQU networks is clear shown by our constructive proofs: by using one hidden layer, a ReQU network can only recover quadratic polynomials; by using $n$ hidden layers, a ReQU network can recover polynomials of degree up to $\mathcal{O}(2^n)$ exactly. The ReQU networks we built for approximating smooth functions all have a tree-like structure, and sparsely connected. This may give some hints on how to design appropriate structures of neural networks for some practical applications.

We have shown that for approximating smooth functions, ReQU networks are superior to ReLU networks in terms of approximation error. In practical applications, the functions to be approximated may have different kinds of non-smoothness, which are problem dependent. The training method is another important issue that affects the application of neural networks. We will continue our study in these directions. In particular, we will study the approximation error of piecewise smooth functions with deep ReQU networks, and investigate whether those popular training methods proposed to train ReLU networks are efficient for training RePU networks. Meanwhile, we will try deep RePU networks on some practical problems where the underlying functions are smooth, e.g. minimum action methods for large PDE system[46], PDEs with random coefficients[47], and moment closure problem in complex fluid [48] and turbulence modeling[49], etc.

## Acknowledgements

This work was partially supported by China National Program on Key Basic Research Project 2015CB856003, NNSFC Grant 11771439 and 91852116.

## Reference

## References

[1] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics 5 (4) (1943) 115–133. doi:10.1007/BF02478259.

[2] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Computation 18 (7) (2006) 1527–1554. doi:10.1162/neco.2006.18.7.1527.

[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Advances in Neural Information Processing Systems, 2007, pp. 153–160.

[4] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[5] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, T. Sainath, Deep neural networks for acoustic modeling in speech recognition, IEEE Signal Process. Mag. 29.

[6] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.

[7] L. Zhang, J. Han, H. Wang, R. Car, W. E, Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics, Phys. Rev. Lett. 120 (14) (2018) 143001. doi:10.1103/PhysRevLett.120.143001.

[8] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, PNAS 115 (34) (2018) 8505–8510. doi:10.1073/pnas.1718942115.

[9] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signal Systems 2 (4) (1989) 303–314. doi:10.1007/BF02551274.

[10] H. N. Mhaskar, Neural networks for optimal approximation of smooth and analytic functions, Neural Computation 8 (1) (1996) 164–177. doi:10.1162/neco.1996.8.1.164.

[11] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195. doi:10.1017/S0962492900002919.

[12] O. Delalleau, Y. Bengio, Shallow vs. deep sum-product networks, in: NIPS, 2011, p. 9.

[13] M. Telgarsky, Representation benefits of deep feedforward networks, ArXiv150908101 Cs.

[14] R. Eldan, O. Shamir, The power of depth for feedforward neural networks, JMLR Workshop Conf. Proc. 49 (2016) 1–34.

[15] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the 14 Th International Conference on Artificial Intelligence and Statistics, Vol. 15, JMLR, Fort Lauderdal, 2011, pp. 315–323.

[16] S. Liang, R. Srikant, Why deep neural networks for function approximation?, ArXiv161004161 CsarXiv:1610.04161.

[17] M. Telgarsky, Benefits of depth in neural networks, in: JMLR: Workshop and Conference Proceedings, Vol. 49, 2016, pp. 1–23.

[18] D. Yarotsky, Error bounds for approximations with deep ReLU networks, Neural Netw. 94 (2017) 103–114. doi:10.1016/j.neunet.2017.07.002.

[19] P. Petersen, F. Voigtlaender, Optimal approximation of piecewise smooth functions using deep ReLU neural networks, Neural Netw. 108 (2018) 296–330. doi:10.1016/j.neunet.2018.08.019.

[20] W. E, Q. Wang, Exponential convergence of the deep neural network approximation for analytic functions, Sci. China Math. 61 (10) (2018) 1733–1740. doi:10.1007/s11425-018-9387-x.

[21] S. A. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Dokl Akad Nauk SSSR 148 (5) (1963) 1042–1045.

[22] H.-J. Bungartz, M. Griebel, Sparse grids, Acta Numer. 13 (2004) 1–123.

[23] M. Griebel, J. Hamaekers, Sparse grids for the Schrödinger equation, Math. Model. Numer. Anal. 41 (2) (2007) 215–247.

[24] J. Shen, L. Wang, Sparse spectral approximations of high-dimensional problems based on hyperbolic cross, SIAM J Numer Anal 48 (4) (2010) 1087–1109.

[25] T. Gerstner, M. Griebel, Numerical integration using sparse grids, Numer. Algorithms 18 (3) (1998) 209–232. doi:10.1023/A:1019129717644.

[26] V. Barthelmann, E. Novak, K. Ritter, High dimensional polynomial interpolation on sparse grids, Adv. Comput. Math. 12 (4) (2000) 273–288.

[27] J. Shen, L.-L. Wang, H. Yu, Approximations by orthonormal mapped Chebyshev functions for higher-dimensional problems in unbounded domains, J. Comput. Appl. Mathemaitcs 265 (2014) 264–275.

[28] H. J. Bungartz, An adaptive Poisson solver using hierarchical bases and sparse grids, in: Iterative Methods in Linear Algebra, Amsterdam: North-Holland, Brussels, Belgium, 1992, pp. 293–310.

[29] J. Shen, H. Yu, Efficient spectral sparse grid methods and applications to high-dimensional elliptic problems, SIAM J. Sci. Comput. 32 (6) (2010) 3228–3250.

[30] J. Shen, H. Yu, Efficient spectral sparse grid methods and applications to high-dimensional elliptic equations II: Unbounded domains, SIAM J. Sci. Comput. 34 (2) (2012) 1141–1164.

[31] Z. Wang, Q. Tang, W. Guo, Y. Cheng, Sparse grid discontinuous Galerkin methods for high-dimensional elliptic equations, J. Comput. Phys. 314 (2016) 244–263. doi:10.1016/j.jcp.2016.03.005.

[32] Z. Rong, J. Shen, H. Yu, A nodal sparse grid spectral element method for multi-dimensional elliptic partial differential equations, Int. J. Numer. Anal. Model. 14 (4-5) (2017) 762–783.

[33] H. Yserentant, The hyperbolic cross space approximation of electronic wavefunctions, Numer. Math. 105 (4) (2007) 659–690. doi:10.1007/s00211-006-0038-x.

[34] G. Avila, T. Carrington, Solving the Schroedinger equation using Smolyak interpolants, J. Chem. Phys. 139 (13) (2013) 134114. doi:10.1063/1.4821348.

[35] J. Shen, Y. Wang, H. Yu, Efficient spectral-element methods for the electronic Schrödinger equation, in: J. Garcke, D. Pflüger (Eds.), Sparse Grids and Applications - Stuttgart 2014, Lecture Notes in Computational Science and Engineering, Springer International Publishing, 2016, pp. 265–289.

[36] C. Schwab, R. Todor, Sparse finite elements for stochastic elliptic problems – higher order moments, Computing 71 (1) (2003) 43–63. doi:10.1007/s00607-003-0024-4.

[37] F. Nobile, R. Tempone, C. Webster, A sparse grid stochastic collocation method for partial differential equations with random input data, SIAM J. Numer. Anal. 46 (5) (2008) 2309–2345. doi:10.1137/060663660.

[38] F. Nobile, L. Tamellini, F. Tesei, R. Tempone, An adaptive sparse grid algorithm for elliptic pdes with lognormal diffusion coefficient, in: J. Garcke, D. Pflüger (Eds.), Sparse Grids and Applications - Stuttgart 2014, Vol. 109, Springer International Publishing, Cham, 2016, pp. 191–220. doi:10.1007/978-3-319-28262-6_8.

[39] H. Montanelli, Q. Du, New error bounds for deep ReLU networks using sparse grids, SIAM J. Math. Data Sci. 1 (1) (2019) 78–92.

[40] J. He, L. Li, J. Xu, C. Zheng, Relu deep neural networks and linear finite elements, ArXiv180703973 MatharXiv:1807.03973.

[41] P. Petrushev, Approximation by ridge functions and neural networks, SIAM J. Math. Anal. 30 (1) (1998) 155–189. doi:10.1137/S0036141097322959.

[42] W. E, B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12. doi:10.1007/s40304-018-0127-z.

[43] J. Shen, T. Tang, L.-L. Wang, Spectral Methods : Algorithms, Analysis and Applications, Springer, 2011.

[44] I. H. Sloan, H. Wozniakowski, When are quasi-Monte Carlo algorithms efficient for high dimensional integrals?, J. Complex. 14 (1) (1998) 1–33. doi:10.1006/jcom.1997.0463.

[45] X. Wang, I. Sloan, Why are high-dimensional finance problems often of low effective dimension?, SIAM J. Sci. Comput. 27 (1) (2005) 159–183. doi:10.1137/S1064827503429429.

[46] X. Wan, H. Yu, A dynamic-solver-consistent minimum action method: With an application to 2D Navier-Stokes equations, Journal of Computational Physics 331 (2017) 209–226. doi:10.1016/j.jcp.2016.11.019.

[47] E. Musharbash, F. Nobile, T. Zhou, Error analysis of the dynamically orthogonal approximation of time dependent random PDEs, SIAM J. Sci. Comput. 37 (2) (2015) A776–A810. doi:10.1137/140967787.

[48] H. Yu, G. Ji, P. Zhang, A nonhomogeneous kinetic model of liquid crystal polymers and its thermodynamic closure approximation, Commun. Comput. Phys. 7 (2) (2010) 383.

[49] G. L. Mellor, T. Yamada, Development of a turbulence closure model for geophysical fluid problems, Rev. Geophys. 20 (4) (1982) 851–875. doi:10.1029/RG020i004p00851.