PowerNet: Efficient Representations of Polynomials and Smooth Functions by Deep Neural Networks with Rectified Power Units

Bo Li\textsuperscript{b,a,1}, Shanshan Tang\textsuperscript{b,a,1}, Haijun Yu\textsuperscript{a,b,c,*}

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

Abstract

Deep neural network with rectified linear units (ReLU) is getting more and more popular recently. However, the derivatives of the function represented by a ReLU network are not continuous, which limit the usage of ReLU network to situations only when smoothness is not required. In this paper, we construct deep neural networks with rectified power units (RePU), which can give better approximations for smooth functions. Optimal algorithms are proposed to explicitly build neural networks with sparsely connected RePUs, which we call PowerNets, to represent polynomials with no approximation error. For general smooth functions, we first project the function to their polynomial approximations, then use the proposed algorithms to construct corresponding PowerNets. Thus, the error of best polynomial approximation provides an upper bound of the best RePU network approximation error. For smooth functions in higher dimensional Sobolev spaces, we use fast spectral transforms for tensor-product grid and sparse grid discretization to get polynomial approximations. Our constructive algorithms show clearly a close connection between spectral methods and deep neural networks: a PowerNet with \( n \) layers can exactly represent polynomials up to degree \( s^n \), where \( s \) is the power of RePUs. The proposed PowerNets have potential applications in the situations where high-accuracy is desired or smoothness is required.

Keywords: deep neural network, rectified linear unit, rectified power unit, sparse grid, PowerNet

1. Introduction

Artificial neural network (ANN) has been a hot research topic for several decades. Deep neural network (DNN), a special class of ANN with multiple hidden layers, is getting more and more popular recently. Since 2006, when efficient training methods were introduced by Hinton et al \cite{1}, DNNs have brought significant improvements in several challenging problems including image classification, speech recognition, computational chemistry and numerical solutions of high-dimensional partial differential equations, see e.g. \cite{2, 3, 4, 5, 6}, and references therein.

The success of ANNs rely on the fact that they have good representation power. Actually, the universal approximation property of neural networks is well-known: neural networks with one hidden layer of continuous/monotonic sigmoid activation functions are dense in continuous function space \( C([0,1]^d) \) and \( L^1([0,1]^d) \), see e.g. \cite{7, 8, 9} for different proofs in different settings. Actually, for neural network with non-polynomial \( C^\infty \) activation functions, the upper bound of approximation error is of spectral type even using only one-hidden layer, i.e. error rate \( \varepsilon = n^{-k/d} \) can be obtained theoretically for approximation functions in Sobolev space \( W^k([-1,1]^d) \), where \( d \) is the number of dimensions, \( n \) is the number of hidden nodes in the neural network\cite{10}. It is believed that one of the basic reasons behind the success of DNNs is the fact that deep neural networks have broader scopes of representation than shallow ones. Recently, several works have demonstrated or proved this in different settings. For example, by using the composition function argument, Poggio et al \cite{11} showed that deep networks can avoid...
the curse of dimensionality for an important class of problems corresponding to compositional functions. In the general function approximation aspect, it has been proved by Yarotsky [12] that DNNs using rectified linear units (abbr. ReLU, a non-smooth activation function defined as $\sigma_1(x) := \max(0, x)$) need at most $O(\epsilon^{-d}(\log |\epsilon| + 1))$ units and nonzero weights to approximation functions in Sobolev space $W^{k,\infty}([-1, 1]^d)$ within $\epsilon$ error. This is similar to the results of shallow networks with one hidden layer of and nonzero weights to approximation functions in Sobolev space (abbr. ReLU, a non-smooth activation function defined as $\sigma = \max(0, x)$) general function approximation aspect, it has been proved by Yarotsky [12] that DNNs using rectified linear units (ReLU) to high-order finite element methods. In this work with DNNs to high-order finite element methods.

A basic fact used in the error estimate given in [12] and [13] is that $x^2, xy$ can be approximated by a ReLU network with $\mathcal{O}(\log |\epsilon|)$ layers. To remove this approximation error and the extra factor $\mathcal{O}(\log |\epsilon|)$ in the size of neural networks, we proposed to use rectified power units (RePU) to construct exact neural network representations of polynomials [15]. The RePU function is defined as

$$
\sigma_s(x) = \begin{cases} 
  x^s, & x \geq 0, \\
  0, & x < 0,
\end{cases}
$$

where $s$ is a non-negative integer. When $s = 1$, we have the Heaviside step function; when $s = 1$, we have the commonly used ReLU function $\sigma_1$. We call $\sigma_2, \sigma_3$ rectified quadratic unit (ReQU) and rectified cubic unit (ReCU) for $s = 2, 3$, respectively. Note that, some pioneering works have been done by Mhaskar and his coworkers (see e.g. [17], [18]) to give a theoretical upper bound of DNN function approximations by converting splines into RePU DNNs. However, for very smooth functions, their constructions of neural network are not optimal and meanwhile are not numerically stable. The error bound obtained is quasi-optimal due to an extra $\log(k)$ factor, where $k$ is related to the smoothness of the underlying functions. The extra $\log(k)$ factor is removed in our earlier work [16] by introducing some explicit optimal and stable constructions of ReQU networks to exactly represent polynomials. In this paper, we extend the results to deep networks using general RePUs with $s \geq 2$.

Comparing with other two constructive approaches (The Qin Jiushao algorithm and the first-composition-then-combination method used in [17], [18], etc), our constructions of RePU neural networks to represent polynomials are optimal in the numbers of network layers and hidden nodes. To approximate general smooth functions, we first approximate the function by its best polynomial approximation, then convert the polynomial approximation into a RePU network with optimal size. The conclusion of algebraic convergence for $W^{k,2}$ functions and exponential convergence for analytic functions then follows straightforward. For multi-dimensional problems, we use the concept of sparse grid to improve the error estimate of neural networks and lessen the curse of dimensionality.

The main advantage of the ReLU function is that ReLU DNNs are relatively easier to train than DNNs using other analytic sigmoidal activation units in traditional applications. The latter ones have well-known severe gradient vanishing phenomenon. However, ReLU networks have some limitations. E.g., due to the fact that the derivatives of a ReLU network function are not continuous, ReLU networks are hard to train when the loss function contains derivatives of the network, thus functions with higher-order smoothness are desired. Such an example is the deep Ritz method solving partial differential equations (PDEs) recently developed by E and Yu [19], where ReQU networks are used.

The remain part of this paper is organized as follows. In Section 2 we first show how to realize univariate polynomials and approximate smooth functions using RePU networks. Then we construct RePU network realization of multivariate polynomials and general multivariate smooth functions in Section 3, with extensions to high-dimensional functions in sparse space given in Subsection 3.3. A short summary is given in Section 4.

## 2. Approximation of univariate smooth functions

We first introduce notations. Denote by $\mathbb{N}$ the set of all positive integer, $\mathbb{N}_0 := \{0\} \cup \mathbb{N}$, $Z_n := \{0, 1, \ldots, n - 1\}$ for $n \in \mathbb{N}$.
Definition 1. We define a neural network $\Phi$ with input of dimension $d \in \mathbb{N}$, number of layer $L \in \mathbb{N}$ as a matrix-vector sequence
\[
\Phi = ((A_1, b_1), \cdots, (A_L, b_L)),
\] (2.1)
where $A_k, k = 1, \ldots, L$ are $N_k \times N_{k-1}$ matrices, $b_k \in \mathbb{R}^{N_k \times 1}$ are vectors called bias, $N_0 = d$ and $N_1, \cdots, N_L \in \mathbb{N}$.

Definition 2. If $\Phi$ is a neural network defined by (2.1), and $\rho : \mathbb{R} \to \mathbb{R}$ is an arbitrary activation function, then define the neural network function
\[
R_\rho(\Phi) : \mathbb{R}^d \to \mathbb{R}^{N_L}, \quad R_\rho(\Phi)(x) = x_L,
\] (2.2)
where $x_L = R_\rho(\Phi)(x)$ is defined as
\[
\begin{aligned}
x_0 &:= x, \\
x_k &:= \rho(A_k x_{k-1} + b_k), \quad k = 1, 2, \ldots, L-1, \\
x_L &:= A_L x_{L-1} + b_L.
\end{aligned}
\] (2.3)
Here we denote vector variables $x_k \in \mathbb{R}^{N_k}$ by bold letters and use the definition
\[
\rho(y) := (\rho(y^1), \cdots, \rho(y^m))^T, \quad \forall \ y = (y^1, \cdots, y^m)^T \in \mathbb{R}^m.
\]

We use three quantities to measure the complexity of a neural network $\Phi$: number of layers $L(\Phi)$, number of nodes (i.e. activation units) $N(\Phi)$, and number of nonzero weights $M(\Phi)$, which are $L, \sum_{k=1}^{L-1} N_k(\Phi)$ and $\sum_{k=1}^{L} M_k(\Phi)$, respectively. For the neural network defined in (2.1), $N_k(\Phi) := N_k$, $k = 0, \ldots, L$ are the dimensions of $x_k$, and $M_k(\Phi) := \|A_k\|_0 + \|b_k\|_0$ (for $k = 1, \ldots, L$) is the number of nonzero weights in the $k$-th affine transformation. Note that, in this paper, we define $L$ as the layers of affine transformations defined in (2.3). We also call $x_0$ the input layer, $x_L$ the output layer, and $x_k, k = 1, \ldots, L-1$ hidden layers. So, there are $L-1$ hidden layers, which is the number of layers of activation units.

Definition 3. We define $\Pi_m^m_{d,N,L}$ as the collection of all neural networks of input dimension $d$, output dimension $m$ with at most $N$ neurons arranged in $L$ layers, i.e.
\[
\Pi_m^m_{d,N,L} := \left\{ \Phi = ((A_1, b_1), \cdots, (A_L, b_L)) \mid A_k \in \mathbb{R}^{N_k \times N_{k-1}}, b_k \in \mathbb{R}^{N_k \times 1}, \text{for } k=1,\ldots,L; N_0 = d, N_L = m, \sum_{k=1}^{L} N_k = N \right\}.
\] (2.4)

For given activation function $\rho$, we further define
\[
\Pi_m^m_{d,N,L,\rho} := \left\{ R_\rho(\Phi) \mid \Phi \in \Pi_m^m_{d,N,L} \right\}.
\] (2.5)

To construct complex networks from simple ones, we first introduce several network compositions.

Definition 4. Let $L_1, L_2 \in \mathbb{N}$ and $\Phi^1 = ((A_1^1, b_1^1), \cdots, (A_{L_1}^1, b_{L_1}^1))$, $\Phi^2 = ((A_1^2, b_1^2), \cdots, (A_{L_2}^2, b_{L_2}^2))$ be two neural networks such that the input layer of $\Phi^1$ has the same dimension as the output layer of $\Phi^2$. We define the the concatenation of $\Phi^1$ and $\Phi^2$ as
\[
\Phi^2 \circ \Phi^1 := ((A_1^1, b_1^1), \cdots, (A_{L_2}^2, b_{L_2}^2), (A_1^{L_1}, A_1^{L_1} b_{L_1}^1 + b_1^1), (A_2^{L_1}, b_2^2), \ldots, (A_{L_2}^{L_1}, b_{L_2}^2)).
\] (2.6)

By the definition, we have
\[
R_{\sigma_{\rho}}(\Phi^2 \circ \Phi^1) = R_{\sigma_{\rho}}(\Phi^2) \circ R_{\sigma_{\rho}}(\Phi^1) =: R_{\sigma_{\rho}}(\Phi^2) \circ R_{\sigma_{\rho}}(\Phi^1),
\]
\[
L(\Phi^2 \circ \Phi^1) = L(\Phi^1) + L(\Phi^2) - 1, \quad N(\Phi^2 \circ \Phi^1) = N(\Phi^1) + N(\Phi^2).
\]
Definition 5. Let $\Phi^1 = ((A_1^{1}, b_1^{1}), \ldots, (A_L^{1}, b_L^{1}))$, $\Phi^2 = ((A_1^{2}, b_1^{2}), \ldots, (A_L^{2}, b_L^{2}))$ be two neural networks both with $L \in \mathbb{N}$ layers. Suppose the input dimensions of the two networks are $d_1, d_2$ respectively. We define the parallelization of $\Phi^1$ and $\Phi^2$ as

$$
\Phi^1 \vee \Phi^2 := ((\bar{A}_1, \tilde{b}_1), \ldots, (\bar{A}_L, \tilde{b}_L)),
$$

where

$$
\bar{A}_i = \begin{bmatrix} A_i^1 & 0 \\ 0 & A_i^2 \end{bmatrix}, \quad \tilde{b}_i = \begin{bmatrix} b_i^1 \\ b_i^2 \end{bmatrix}, \quad \text{and} \quad \bar{A}_i = \begin{bmatrix} A_i^1 & 0 \\ 0 & A_i^2 \end{bmatrix}, \quad \tilde{b}_i = \begin{bmatrix} b_i^1 \\ b_i^2 \end{bmatrix}, \quad \text{for} \quad 1 < i \leq L.
$$

Here $\bar{A}_i, i = 1, 2$ are formed from $A_i^1, i = 1, 2$ correspondingly, by padding zero columns in the end to one of them such that they have same number of columns. Obviously, $\Phi^1 \vee \Phi^2$ is a neural network with $\max(d_1, d_2)$-dimensional input and $L$ layers. We have the relationship

$$
R_{\sigma_l}(\Phi^1 \vee \Phi^2) = (R_{\sigma_2}(\Phi^1), R_{\sigma_2}(\Phi^2)),
$$

$$
N(\Phi^1 \vee \Phi^2) = N(\Phi^1) + N(\Phi^2), \quad M(\Phi^1 \vee \Phi^2) = M(\Phi^1) + M(\Phi^2).
$$

For $\Phi^1, \Phi^2$ defined as above but not necessarily have same dimensions of input, we define the tensor product of $\Phi^1$ and $\Phi^2$ as

$$
\Phi^1 \otimes \Phi^2 := ((\bar{A}_1, \tilde{b}_1), \ldots, (\bar{A}_L, \tilde{b}_L)),
$$

where

$$
\bar{A}_i = \begin{bmatrix} A_i^1 & 0 \\ 0 & A_i^2 \end{bmatrix}, \quad \tilde{b}_i = \begin{bmatrix} b_i^1 \\ b_i^2 \end{bmatrix}, \quad \text{for} \quad 1 < i \leq L.
$$

Obviously, $\Phi^1 \otimes \Phi^2$ is a $L$-layer neural network with $N_0(\Phi^1) + N_0(\Phi^2)$ dimensional input and $N_L(\Phi^1) + N_L(\Phi^2)$ dimensional output. We have the relationship

$$
R_{\sigma_l}(\Phi^1 \otimes \Phi^2) = (R_{\sigma_2}(\Phi^1), R_{\sigma_2}(\Phi^2)),
$$

$$
N_k(\Phi^1 \otimes \Phi^2) = N_k(\Phi^1) + N_k(\Phi^2), \quad \forall \, k = 0, \ldots, L, \quad M_k(\Phi^1 \otimes \Phi^2) = M_k(\Phi^1) + M_k(\Phi^2) \quad \forall \, k = 1, \ldots, L.
$$

2.1. Basic properties of RePU networks

Our analyses rely upon the fact: $x, x^2, \ldots, x^s$ and $xy$ can all be realized by a one-hidden-layer $\sigma_s$ neural network with a few number of coefficients, which is presented in the following lemma.

Lemma 1. The monomials $x^n, 1 \leq n \leq s$ can be exactly represented by neural networks with one hidden layer of a finite number of $\sigma_s(x)$ activation nodes. More precisely:

(i) For $s = n$, the monomial $x^n$ can be realized exactly using a $\sigma_s$ network having one hidden layer with two nodes as following.

$$
x^n = \gamma_0^T \sigma_s(a_0x), \quad \gamma_0 = \begin{bmatrix} 1 \\ (-1)^s \end{bmatrix}, \quad a_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.
$$

2.1. Basic properties of RePU networks

Our analyses rely upon the fact: $x, x^2, \ldots, x^s$ and $xy$ can all be realized by a one-hidden-layer $\sigma_s$ neural network with a few number of coefficients, which is presented in the following lemma.

Lemma 1. The monomials $x^n, 1 \leq n \leq s$ can be exactly represented by neural networks with one hidden layer of a finite number of $\sigma_s(x)$ activation nodes. More precisely:

(i) For $s = n$, the monomial $x^n$ can be realized exactly using a $\sigma_s$ network having one hidden layer with two nodes as following.

$$
x^n = \gamma_0^T \sigma_s(a_0x), \quad \gamma_0 = \begin{bmatrix} 1 \\ (-1)^s \end{bmatrix}, \quad a_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.
$$

Correspondingly, the neural network is defined as

$$
\Phi_{n_0}^1 = \{(a_0, 0), (\gamma_0^T, 0)\}.
$$

A graph representation of $\Phi_{n_0}^1$ is sketched in Fig. 2a.
(ii) For \(1 \leq n \leq s\), the monomial \(x^n\) can be realized exactly using a \(\sigma_s\) network having only one hidden layer

\[
x^n = \gamma^T_{1,n} \sigma_s(a_1 x + \beta_1) + \lambda_{0,n}, \quad n = 1, \ldots, s - 1,
\]

where

\[
\begin{bmatrix} a_0 \\ \vdots \\ a_0 \end{bmatrix} \in \mathbb{R}^{2s \times 1}, \quad \begin{bmatrix} b_1 a_0 \\ \vdots \\ b_s a_0 \end{bmatrix} \in \mathbb{R}^{2s \times 1}, \quad \begin{bmatrix} \lambda_{1,n} \gamma_0 \\ \vdots \\ \lambda_{s,n} \gamma_0 \end{bmatrix} \in \mathbb{R}^{2s \times 1},
\]

(2.12)

Here \(b_1, \ldots, b_s\) are distinct points in \(\mathbb{R}\). We suggest to use (2.20) - (2.24) for \(s \leq 6\) and (2.18) for \(s > 6\). \(\lambda_{0,n}, \lambda_{1,n}, \ldots, \lambda_{s,n}\) are calculated by (2.17). The neural network is defined as

\[
\Phi_{\text{mo},n}^2 = ((a_1, \beta_1), (\gamma^T_{1,n}, \lambda_{0,n})).
\]

(2.13)

A graph representation of \(\Phi_{\text{mo},n}^2\) is sketched in Fig. 2b. Note that, when \(n = 0\), we have a trivial realization:

\[
\alpha_1 = \beta_1 = \gamma_{1,0} = 0, \quad \gamma_{0,0} = 1. \quad \text{When} \quad n = s, \quad \text{the implementation in (i) is more efficient.}
\]

When \(n = 1\), we obtain the network realization of identity function \(\Phi_{\text{idk}}:= \Phi_{\text{mo},1}^2\).

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

\[
\rho_s(x):= \sigma_s(x) + (-1)^s \sigma_s(-x) = \gamma^T_{0} \sigma_s(a_0 x).
\]

(2.14)

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

\[
\lambda_0 + \sum_{k=1}^{s} \lambda_k \rho_s(x + b_k) = \lambda_0 + \sum_{k=1}^{s} \lambda_k \left( \sum_{j=0}^{s} C^j_k b_{k}^{-j} x^j \right) = \lambda_0 + \sum_{j=0}^{s} C^j_0 \left( \sum_{k=1}^{s} \lambda_k b_{k}^{-j} \right) x^j,
\]

(2.15)

where \(\lambda_0, \lambda_k, b_k, k = 1, 2, \ldots, s\) are parameters to be determined. \(C^j_k, i = 0, 1, \ldots, s\) are binomial coefficients. Identify the above expression with a polynomial of degree does not exceed \(s\), i.e. \(\sum_{k=0}^{s} d_k x^k\), we obtain the following linear system

\[
D_{s+1} \mathbf{A}_s := \begin{bmatrix} 1 & 1 \cdots & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ b_1^{s-i} & b_2^{s-i} \cdots & b_s^{s-i} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ b_1 & b_2 \cdots & b_s \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_s \end{bmatrix} = \begin{bmatrix} d_s(C_s^0)^{-1} \\ \vdots \\ d_0(C_0^0)^{-1} \end{bmatrix},
\]

(2.16)

where the top-left \(s \times s\) sub-matrix of \(D_{s+1}\) is a Vandermonde matrix \(V_s\), which is invertible as long as \(b_k, k = 1, 2, \ldots, s\) are distant. The choices of \(b_k\) are discussed later in Remark 1. Denote \(\lambda_s = [\lambda_1, \ldots, \lambda_s, \lambda_0]^T, \quad b = [b_1^s, \ldots, b_s^s]^T, \quad d = [d_0, \ldots, d_0]^T\). We have

\[
D_{s+1}^{-1} = \begin{bmatrix} V_s^{-1} & 0 \\ b^T \end{bmatrix}, \quad D_{s+1}^{-1} = \begin{bmatrix} V_s^{-1} & 0 \\ b^TV_s^{-1} \end{bmatrix}.
\]

then

\[
\lambda_s = \begin{bmatrix} V_s^{-1} \\ -b^TV_s^{-1} \end{bmatrix} \text{diag}(C_s^0, (C_s^{-1})^{-1}, \ldots, (C_0^0)^{-1})d.
\]

(2.17)

To represent \(x^n \) (\(1 \leq n \leq s\)), we have \(d = e_{s+1}^{n+1} \) in (2.17), where \(e_{s+1}^{n+1} = [\delta_{s+1,1}, \ldots, \delta_{s+1,k}]^T\) and \(\delta_{i,k}\) is the Kronecker delta function.
Remark 1. The inverse of Vandermonde matrix will inevitably be involved in the solution of (2.17), which make the formula (2.11) difficult to use for large $s$ due to the geometrically growth of the condition number of the Vandermonde matrix [21,22,20]. The condition number of the $s \times s$ Vandermonde matrices with three different choices of symmetric nodes are given in Figure 1. The three choices for symmetric nodes are Chebyshev nodes

$b_k = \cos\left(\frac{k-1}{s-1}\pi\right), \quad k = 1, \ldots, s,$ \hspace{1cm} (2.18)

equidistant points

$b_k = 1 - 2 \frac{k-1}{s-1}, \quad k = 1, \ldots, s,$ \hspace{1cm} (2.19)

and numerically calculated optimal nodes. The counterparts of these three different choices for non-negative nodes are also depicted in Figure 1. Most of the results are from [20]. For large $s$ the numerical procedure to calculate the optimal nodes may not succeed. But the growth rates of the $l_\infty$ condition number of Vandermonde matrices using Chebyshev nodes on $[-1,1]$ is close to the optimal case, so we use Chebyshev nodes (2.18) for large $s$. For smaller
values of $s$, we use numerically calculated optimal nodes, which are given for $2 \leq s \leq 6$ in [21]:

\[
\begin{align*}
    b_1 &= -b_2 = 1, & s = 2 \\
    b_1 &= -b_3 = \sqrt{3/2} = 1.2247448714, & b_2 = 0, \\
    b_1 &= -b_4 = 1.2228992744, & b_2 = 0.5552395908, \\
    b_1 &= -b_5 = 1.2001030479, & b_2 = -b_4 = 0.8077421768, \\
    b_1 &= -b_6 = 1.1601101028, & b_2 = -b_5 = 0.9771502216, \\
    & b_3 = 0, & s = 5 \\
    & b_3 = 0.3788765912, & s = 6
\end{align*}
\]  

Note that, in some special cases, if non-negative nodes are used, the number of activation functions in the network construction can be reduced. However, due to the fact that the condition number in this case is larger than the case with symmetric nodes, we will not consider the use of all non-negative nodes in this paper.

Based on Lemma 1 one can easily obtain following results.

**Corollary 1.1.** A univariate polynomial with degree up to $s$ can be exactly represented by neural networks with one hidden layer of $2s$ activation nodes. More precisely, by [21,71], we have

\[
\sum_{j=0}^{s} d_j x^j = d_0 + \sum_{j=1}^{s} d_j \left( y_{1,j} \sigma_1(x + \beta_1) + \lambda_0,j \right) = \tilde{y}_3 \sigma_1(x + \beta_1) + \tilde{c}_1, \tag{2.25}
\]

where $\tilde{y}_3 = \sum_{j=1}^{s} d_j y_{1,j}$, $\tilde{c}_1 = d_0 + \sum_{j=1}^{s} d_j \lambda_0,j$. The corresponding neural network is defined as

\[
\Phi_{pm}^1(d) = ([x_1, \beta_1], (\tilde{y}_3, \tilde{c}_1)), \tag{2.26}
\]

where $d = [d_1, \ldots, d_s, d_0]^T$. A graph representation of $\Phi_{pm}^1$ is sketched in Fig. 2c.

In the implementation of polynomials, operations of the form $x^n y$ will be frequently involved. Following lemma asserts that $x^n y, 0 \leq n \leq s - 1$ can be realized by using only one hidden layer.

**Lemma 2.** Bivariate monomials $x^n y, 0 \leq n \leq s - 1$ can be realized as a linear combination of at most $u_n$ activation units of $\sigma_1(\cdot)$ as

\[
x^n y = \gamma_{2,n}^T \sigma_2(\alpha_{2,n,1} x + \alpha_{2,n,2} y + \beta_{2,n}), \quad n = 0, 1, \ldots, s - 1, \tag{2.27}
\]

where $\alpha_{2,n,1}, \alpha_{2,n,2}, \beta_{2,n}, \gamma_{2,n} \in \mathbb{R}^{u_n \times 1}$, $u_n = 2(n + 1)(s - n)$. A particular formula is given by (4.8) in the appendix section. The corresponding neural network is defined as

\[
\Phi_{bm,n}^1 = \{(\alpha_{2,n,1}, \alpha_{2,n,2}, \beta_{2,n}), (\gamma_{2,n}^T, 0)\}. \tag{2.28}
\]

A graph representation of $\Phi_{bm,n}^1$ is sketched in Fig. 2d. Obviously, the numbers of nonzero weights in the first layer and second layer affine transformation are $3u_n$ and $u_n$ correspondingly.

The proof of Lemma 2 is lengthy. We put it in the appendix section.

**Corollary 2.1.** A polynomial of the form $\sum_{k=0}^{s-1} x^k y_k$ can be realized as a linear combination of at most $w$ activation units of $\sigma_1(\cdot)$ as

\[
\Phi_{pm}^1 = \Phi_0 \circ \{\Phi_{mo,1}^2 \lor \Phi_{bm,1}^1 \lor \cdots \lor \Phi_{bm,s-1}^1\}. \tag{2.29}
\]

Here $\Phi_0 = (1, 0)$ with $1 = (1, \ldots, 1)^T \in \mathbb{R}^s$ contains only a linear combination layer. A graph representation of $\Phi_{pm}^1$ is sketched in Fig. 2e. The numbers of nonzero weights in the first layer and second layer affine transformations are at most $3w$ and $w$ correspondingly. Here

\[
w = 2s + \sum_{k=1}^{s-1} u_k = 2s + \sum_{k=1}^{s-1} 2(k + 1)(s - k) = \frac{1}{3} (s^3 + 3s^2 + 2s)
\]
Figure 2: Some shallow neural networks used as building bricks of the RePUs DNNs. Here circles represent hidden nodes, squares represent input, output and intermediate variables, A “+” sign inside a circle or a square represent a nonzero bias.
2.2. Optimal realizations of polynomials by RePU networks with no error

The basic properties of \( \sigma \), given in Lemma 1 and Lemma 2, can be used to construct neural network representation of any monomial and polynomial. We first present the results of monomial.

For \( x^n \) with \( 1 \leq n \leq s \), by Lemma 1, the number of layers, hidden units and nonzero weights required in a \( \sigma \) network to realize it is no more than 2, 2, 6s + 1, correspondingly. For \( n > s \), we have the following Theorem.

**Theorem 1.** For \( 2 \leq s < n \in \mathbb{N} \), there exist a \( \sigma \) network \( \Phi_{\text{mo}}^s \) with

\[
L(\Phi_{\text{mo}}^s) \leq \lfloor \log_s n \rfloor + 1, \quad N(\Phi_{\text{mo}}^s) \leq \lfloor \log_s n \rfloor ((s + 1)/2 + 2) + 2s \\
M(\Phi_{\text{mo}}^s) \leq (\lfloor \log_s n \rfloor - 1)(u^2 + 3u + 4) + 2su + 4u + 4s + 2, \quad u := (s + 1)/4
\]


to exactly represent the monomial \( x^n \) defined on \( \mathbb{R} \). Here, \( \lfloor x \rfloor \) represents the largest integer not exceeding \( x \), and \( \lceil x \rceil \) represents the smallest integer no less than \( x \), for \( x \in \mathbb{R} \).

**Proof.** 1) For \( n > s, \log_s n \in \mathbb{Z} \), we first express \( n \in \mathbb{N} \) in positional numeral system with radix \( s \) as follows:

\[
n = n_m \cdot s^m + n_{m-1} \cdot s^{m-1} + \cdots + n_1 \cdot s + n_0 =: (n_m \cdots n_1 n_0)_s,
\]

where \( m = \lfloor \log_s n \rfloor, n_j \in \mathbb{Z}_s \) for \( j = 0, \ldots, m - 1 \) and \( 0 \neq n_m \in \mathbb{Z}_s \). Then

\[
x^n = x^{n_m s^m} \cdot x^{n_{m-1} s^{m-1}} \cdots x^{n_1 s} x^{n_0}.
\]

Introducing intermediate variables

\[
\xi_k^{(1)} := x^{\xi_k}, \quad \xi_k^{(2)} := x^{\xi_k}, \quad \text{for } 1 \leq k \leq m+1,
\]

then \( x^n = \xi_{m+1}^{(2)} \) can be calculated iteratively as

\[
\begin{align*}
\xi_k^{(1)} &= x^k, \\
\xi_k^{(2)} &= x^{n_0}, \\
\xi_k^{(1)} &= (\xi_{k-1}^{(1)})^s, \\
\xi_k^{(2)} &= (\xi_{k-1}^{(1)})^{n_{k-1}} \xi_{k-1}^{(2)}, \\
\xi_{m+1}^{(1)} &= (\xi_m^{(1)})^{s^m} \xi_m^{(2)},
\end{align*}
\]

Therefore, to construct a \( \sigma \) neural network expressing \( x^n \), we need to realize three basic operations: \((-)^s \), \((-)^n \) and multiplication. By Lemma 1, each step of iteration (2.33) can be realized by a \( \sigma \) network with one hidden layer. Then the overall neural network to realize \( x^n \) is a concatenation of those one-layer sub-networks. We give the construction process of the neural network as follows.

1. For \( k = 1 \), the first sub-network \( \Phi_1 = (A_1, b_1^1), (A_2, b_2^1) \) is \( \Phi_{\text{mo}}^1 \lor \Phi_{\text{mo}, n_0}^2 \) is constructed according to Lemma 1 as

\[
\begin{align*}
x_0^{(1)} &= x, \\
x_1^{(1)} &= \sigma_s \left( \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} x_0^{(1)} + \begin{bmatrix} 0 \\ b_1 \end{bmatrix} \right) =: \sigma_s \left( A_1 x_0 + b_1^1 \right), \\
x_2^{(1)} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} x_1^{(1)} + \begin{bmatrix} 0 \\ \lambda_{0,n_0} \end{bmatrix} =: A_2 x_1 + b_2^1.
\end{align*}
\]

It is easy to see that the number of nodes in the hidden layer is \( 2(s + 1) \), and the number of non-zeros in \( A_1 \) and \( b_1^1 \) is \( 4s + 2 \).
For $k = 2, \ldots, m$ the sub-network $\Phi^{k} = (A^{k}_{1}, b^{k}_{1}),(A^{k}_{2}, b^{k}_{2}) = \Phi_{mo}^{1} \vee \Phi_{bm,n_{k-1}}$ are constructed as

$$x_{0}^{(k)} = [\xi_{k-1}^{(1)}, \xi_{k-1}^{(2)}]^T,$$
$$x_{1}^{(k)} = \sigma_{x} \left[ \begin{array}{c} \alpha_{0} \\ \alpha_{2,n_{k-1}} \\ \alpha_{2,n_{k-1},2} \end{array} \right] x_{0} + \left[ \begin{array}{c} 0 \\ \beta_{2,n_{k-1}} \end{array} \right] =: \sigma_{x} \left[ A^{k}_{1} x_{0}^{(k)} + b^{k}_{1} \right],$$
$$x_{2}^{(k)} = [\xi_{k}^{(1)}, \xi_{k}^{(2)}]^T = \left[ \begin{array}{c} \gamma_{0}^{T} \\ \gamma_{2,n_{k-1}}^{T} \end{array} \right] x_{1} =: A^{k}_{2} x_{1}^{(k)} + b^{k}_{2}.$$  \hfill (2.35)

The number of nodes in layer $k$ is $2(n_{k-1} + 1)(s - n_{k-1}) + 2$, and the number of non-zeros in $A^{k}_{1}$ and $b^{k}_{1}$ is at most $6(n_{k-1} + 1)(s - n_{k-1}) + 2 \leq 3(s + 1)^{2}/2 + 2$. The number of non-zeros in $A^{k}_{2}$ and $b^{k}_{2}$ is at most $2(n_{k-1} + 1)(s - n_{k-1}) + 2 \leq (s + 1)^{2}/2 + 2$.

For $k = m + 1$, the sub-network $\Phi^{m+1} = (A^{m+1}_{1}, b^{m+1}_{1}),(A^{m+1}_{2}, b^{m+1}_{2}) = \Phi_{bm,n_{m}}$ is constructed as

$$x_{0}^{(m+1)} = [\xi_{m}^{(1)}, \xi_{m}^{(2)}]^T,$$
$$x_{1}^{(m+1)} = \sigma_{x} \left[ \alpha_{2,n_{m-2}} x_{0}^{(m+1)} + \beta_{2,n_{m}} \right] =: \sigma_{x} \left( A^{m+1}_{1} x_{0}^{(m+1)} + b^{m+1}_{1} \right),$$
$$x_{2}^{(m+1)} = x_{m} = \gamma_{2,n_{m}} x_{1}^{(m+1)} =: A^{m+1}_{2} x_{1}^{(m+1)} + b^{m+1}_{2}.$$ \hfill (2.36)

By a straightforward calculation, we get the number of nodes in Layer $m + 1$ is at most $2(n_{m} + 1)(s - n_{m})$, and the number of non-zeros in $A^{m+1}_{2}$ and $b^{m+1}_{2}$ is $2(n_{m} + 1)(s - n_{m}) \leq (s + 1)^{2}/2$.

The whole neural network $\Phi_{mo}^{3}$ is constructed by a concatenation of the all sub-networks, i.e.

$$\Phi_{mo}^{3} = \Phi_{mo}^{m+1} \circ \cdots \circ \Phi_{mo}^{1} = \left( (A^{1}_{1}, b^{1}_{1}),(A^{1}_{2}, b^{1}_{2}) \right), \ldots, (A^{m+1}_{1}, A^{m+1}_{2}, A^{m+1}_{1} b^{m+1}_{1} + b^{m+1}_{2}), \ldots, (A^{1}_{1}, A^{m+1}_{2}, A^{m+1}_{1} b^{m+1}_{1} + b^{m+1}_{2}) \right).$$ \hfill (2.37)

The network structure is sketched in Figure 3. According to the definition, $L(\Phi_{mo}^{3}) = m + 2$. The total number of nodes is given by

$$N(\Phi_{mo}^{3}) = \sum_{k=1}^{m+1} N(\Phi^{k}) = 2(s + 1) + 2(n_{m} + 1)(s - n_{m}) + \sum_{k=2}^{m} (2(n_{k-1} + 1)(s - n_{k-1}) + 2) \leq m(s + 1)^{2}/2 + 2m + 2s.$$
The number of non-zeros is given by

\[
M(\Phi_{mo}^1) = (\|A_1^1\|_0, \|b_1^1\|_0) + (\|A_2^{m+1}\|_0, \|b_2^{m+1}\|_0) + \sum_{k=1}^{m} \|A_1^{k+1}A_2^k\|_0 + \|A_1^{k+1}b_2^k + b_1^{k+1}\|_0
\]

\[
= (4s + 2) + 2(n_m + 1)(s - n_m)
\]

\[
+ [u_0(2s + 3) + 4] + [u_m(u_m - 1) + 3] + \sum_{k=2}^{m-1} u_k(u_k - 1) + 4
\]

\[
\leq (m - 1)(u^2 + 3u + 4) + 2su + 4u + 4s + 2,
\]

where \(u_k = 2(n_k + 1)(s - n_k) \leq u := (s + 1)^2 / 2.\)

2) For \(n > s, \log_2 n = m \in \mathbb{Z},\) we have \(x^n = x^{m}\), which can be realized by a concatenation of \(m\) shallow networks realizing \(x^s\). So the number of layers, nodes and nonzero weights in this network realization is \(m + 1, 2m\) and \(4m\), correspondingly.

\[\Box\]

**Algorithm 1** \textsc{PNet\_Monomial}(s, n)

**Input:** \(n \in \mathbb{N}, 2 \leq s \in \mathbb{N}\)

**Output:** \(\Phi_{mo}\) (with property \(R_{\sigma_1}(\Phi_{mo})(x) = x^n\)).

1. \textbf{if} \(n \leq 1\) \textbf{then}
   2. \textbf{Form} \(\Phi_{mo} = ((\delta_{n,1}, \delta_{n,0}))\).
   3. \textbf{else} \(n = s\) \textbf{then}
   4. \textbf{Form} \(\Phi_{mo} = \Phi_{mo}^1 = ((a_0, 0), (y_0^T, 0))\) with \(y_0, a_0\) defined in (2.9).
   5. \textbf{else if} \(2 \leq n < s\) \textbf{then}
   6. \textbf{Form} \(\Phi_{mo} = \Phi_{mo,n}^1 = ((a_1, \beta_1), (y_{1,n}^T, \lambda_{0,n})\) with \(a_1, \beta_1, y_{1,n}^T, \lambda_{0,n}\) given in Lemma 1(ii).
   7. \textbf{else if} \(n > s\) \textbf{then}
   8. \textbf{Let} \(m = \lceil \log_2 n \rceil\)
   9. \textbf{if} \(2^m = n\) \textbf{then}
   10. \textbf{Form} \(\Phi_{mo} = ((a_0, 0), (a_0y_0^T, 0), \ldots, (a_0y_0^T, 0), (y_0^T, 0))\) with \(m + 1\) layers.
   11. \textbf{else}
   12. \textbf{(1)} \textbf{Take} \((n_m, \ldots, n_1, n_0)\) such that \(n = (n_m \cdots n_1 n_0)_s\),
   13. \textbf{(2)} \textbf{Form} \((A_j^1, b_j^1)\) \((j = 1, 2)\) given in (2.34).
   14. \textbf{(3)} \textbf{Form} \((A_j^k, b_j^k)\) \((j = 1, 2)\) given in (2.35) for \(k = 2, \ldots, m\).
   15. \textbf{(4)} \textbf{Form} \((A_j^{m+1}, b_j^{m+1})\) \((j = 1, 2)\) given in (2.36).
   16. \textbf{(5)} \textbf{Form} \(\Phi_{mo} = \Phi_{mo}^s\) as defined in (2.37).
17. \textbf{return} \(\Phi_{mo}\).

**Remark 2.** It is easy to check that: For any neural network with only one hidden \(\sigma_1\) layer, the corresponding neural network function is a piecewise polynomial of degree \(s\), for any neural network with \(k\) hidden \(\sigma_1\) layers, the corresponding network function is a piecewise polynomial of degree \(s^k\). So \(x^n\) can't be exactly represented by a \(\sigma_1\) neural network with less than \([\log_2 n] - 1\) hidden layers.

**Remark 3.** The detailed procedure presented in Lemma 1 and Theorem 1 is implemented in Algorithm 1. Note that this algorithm generates \(\sigma_1\) DNN to represent monomial \(x^n\) with least(optimal) hidden layers. For large \(n\) and \(s\), the numbers of nodes and nonzero weights in the network is of order \(\mathcal{O}(s^2 \log_2 n)\) and \(\mathcal{O}(s^4 \log_2 n)\), respectively, which are not optimal. To lessen the size of the constructed network for large \(s\), one may implement \((\xi_{k-1})^{n_{k-1}} \cdots \xi_{k-1})\) in (2.33) in two steps: i) implement \(z = (\xi_{k-1})^{n_{k-1}} \cdots \xi_{k-1}\) ii) implement \(z \xi_{k-1}\). According to Lemma 2 and Lemma 3. This will lessen both the number of nodes and the number of nonzero weights in the overall network but will add one more hidden layer. To keep the paper tight, we will not present the detailed implementation of this approach here. Instead, we will describe this approach in the \(\sigma_1\) network realization of polynomials.
Now we consider converting univariate polynomials into $\sigma_s$ networks. If we directly apply Lemma 1 and Theorem 1 to each monomial term in a polynomial of degree $n$ and then combine them together, one would obtain a network of depth $\Theta(\lceil \log_n n \rceil)$ and size at least $\Theta(sn\lceil \log_n n \rceil)$, which is not optimal in terms of network size. Fortunately, there are several other ways to realize polynomials. Next, we first discuss two straightforward constructions. The first one is a direct implementation of Horner's method (also known as Qin Jiushao's algorithm):

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n = a_0 + x\left(a_1 + x(a_2 + x(a_3 + \ldots + x(a_{n-1} + xa_n))\right). \quad (2.38)$$

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

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

Then we have $y_0 = f(x)$. But an iterative implementation of $y_k$ using realizations given in Lemma 1 and Corollary 1.1, from which we see one need to solve some Vandermonde system to obtain the coefficients. The number of nodes is not optimal. Another issue in the latter approach is that one has to calculate the coefficients and then combine them together, one would obtain a $\sigma_s$ neural network with $n$ layers and each hidden layer has $4(s-1)$ activation units.

The second way is the method used by Mhaskar and his coworkers (see e.g. [23, 17]), which is based on the Vandermonde matrix is known has very large condition number for large dimension. A way to avoid solving a Vandermonde system is demonstrate in the proof of Lemma 2. However, from the explicit formula given in (4.5), we see when $n$ is big, large coefficients with different signs coexist, which is deemed to have a large cancellation error. So, lifting the activation function from $\rho_s$ to $\rho_n$ directly is not a numerically stable approach.

Remark 4. The Horner’s method and Mhaskar’s Method have different properties. The first one is optimal in the number of nodes but use too many hidden layers; the latter one is optimal in the number of hidden layers, but the number of nodes is not optimal. Another issue in the latter approach is that one has to calculate the coefficients $c_k, \omega_k, b_k$ in (2.40), which is not an easy task. Note that, when $d = 1$, Proposition 2.1 is equivalent to Lemma 1 and Corollary 1.1 from which we see one need to solve some Vandermonde system to obtain the coefficients. The Vandermonde matrix is known has very large condition number for large dimension. A way to avoid solving a Vandermonde system is demonstrate in the proof of Lemma 2. However, from the explicit formula given in (4.5) - (4.6), we see when $s$ is big, large coefficients with different signs coexist, which is deemed to have a large cancellation error. So, lifting the activation function from $\rho_s$ to $\rho_n$ directly is not a numerically stable approach.

Now, we propose a construction method that avoids the problem of solving large Vandermonde systems. At the same time, the networks we constructed have no very large coefficients.

Consider a polynomial $p(x)$ with degree $n$ greater than $s$. Let $m = \lceil \log_n n \rceil$. We first use a recursive procedure similar to the monomial case to construct a network with minimal layers.

i) Let $n_1 = \lceil n/s \rceil$, we consider $p(x)$ as a polynomial of degree $n_1 s$ by adding zero high degree monomials if

\[ x \to \prod_{i=1}^{r} \sigma_{\rho_i}(x) \]
n_1 s > n. To realize \( p(x) = \sum_{k=0}^{m} a_k x^k \) using a \( \sigma_s \) network, we first divide the summation into \( n_1 \) groups as

\[
p(x) = \sum_{k=0}^{n_1-1} a_k x^k = \sum_{k=0}^{n_1-1} \sum_{j=0}^{s-1} a_{k+j} x^{k+j} + \sum_{j=0}^{s} a_{k(n_1-1)+j} x^{(n_1-1)+j} \quad (a_k = 0 \text{ for } k > n)
\]

\[
= \sum_{k=0}^{n_1-1} \left( (x^s)^k \sum_{j=0}^{s-1} a_{k+j} x^j \right) + (x^s)^{n_1-1} \sum_{j=0}^{s} a_{k(n_1-1)+j} x^j
\]

\[
= \sum_{k=0}^{n_1-1} \sum_{j=0}^{s-1} a_{k+j} x^j y_{1,k},
\]

where

\[
z_1 = x^s, \quad y_{1,k} = \sum_{j=0}^{s-1} a_{k+j} x^j \text{ for } k = 0, \ldots, n_1 - 2, \quad y_{1,n_1-1} = \sum_{j=0}^{s} a_{k(n_1-1)+j} x^j.
\] (2.41)

The above quantities \( \{z_1, y_{1,k}, k = 0, \ldots, n_1 - 1\} \) can be realized by a \( \sigma_s \) network \( \Phi_{n_1}^1 = \Phi_{m_1}^1 \oplus \Phi_{n_2}^1 \in \Pi_{N_1 N_2}^{n_1+1} \) with one hidden layer, where \( \Phi_{n_1}^1 \) is implemented in Fig. 2h. The number of hidden nodes, and numbers of nonzero weights could be as small as

\[
N(\Phi_{n_1}^1) = 2 + 2s, \quad 1 \quad M_1(\Phi_{n_1}^1) = 2 + 4s, \quad 1 \quad M_2(\Phi_{n_2}^1) = 2 + \sum_{k=0}^{n_1-1} (2s + 1), \quad 2
\] (2.42)

where \( 2s+1 \) in the last term means each \( y_{1,k} \) depending on \( 2s \) nodes and 1 shift value. After above procedure, we have reduced the original univariate polynomial of degree \( n \) to a polynomial of degree \( n_1 \). Note that here \( \{y_{1,k}\}, z_1 \) are all variables.

ii) Define \( n_2 = \lceil n_1 / s \rceil \). For the resulting polynomial we can use similar procedure to get

\[
p(x) = \sum_{k=0}^{n_2-1} \sum_{j=0}^{s-1} a_{k+j} x^j y_{1,k+j} = \sum_{k=0}^{n_2-1} \sum_{j=0}^{s-1} \left( \sum_{k=0}^{s-1} a_{k+j} x^j \right) y_{1,k+j}
\]

\[
= \sum_{k=0}^{n_2-1} \sum_{j=0}^{s-1} z_1^j y_{2,k},
\]

where

\[
z_2 = z_1^s, \quad y_{2,k} = \sum_{j=0}^{s-1} z_1^j y_{1,k+j} \text{ for } k = 0, \ldots, n_2 - 1,
\] (2.43)

which, according to Lemma 1 and Lemma 2, can be realized by a neural network \( \Phi_{n_2}^2 \) of only one hidden layer. More precisely,

\[
\Phi_{n_2}^2 = \Phi_{m_2}^2 \oplus \left( \left( \otimes_1 \right)_{k=1}^{n_2} \Phi_{pm_2}^1 \right),
\] (2.44)

where \( \Phi_{pm_2}^1 \) is defined in (2.29), a graph representation is sketched in Fig. 2h. The operator \( \otimes_1 \) is similar to \( \otimes \) but all the sub-nets share one common input, which is taken as the first input of the composited net.

The number of nodes, and numbers of nonzero weights are

\[
N(\Phi_{n_2}^2) = 2 + n_2 \sum_{j=0}^{s-1} u_j = 2 + n_2 w
\]

\[
M_1(\Phi_{n_2}^2) = 2 + 3n_2 \sum_{j=0}^{s-1} u_j = 2 + 3n_2 w \quad M_2(\Phi_{n_2}^2) = 2 + n_2 \sum_{j=0}^{s-1} u_j = 2 + n_2 w
\] (2.46)
iii) For $i = 2, \ldots, m$, repeat similar procedure as ii). Let $n_{i+1} = \lceil n_i/s \rceil$, and using a $\sigma_s$ network $\Phi^{i+1}_a$ with one hidden layer to realize

$$z_{i+1} = z_i^s, \quad y_{i+1,k} = \sum_{j=0}^{s-1} z_i^j y_{i,k+j}, \quad \text{for } k = 0, \ldots, n_{i+1} - 1.$$  \hfill (2.47)

The number of nodes and nonzero weights are similar to the second step with $n_2$ replaced by $n_{i+1}$. The recursive procedure ends at $i = m$. We obtain this conclusion by looking at the base-s form of $n$: $(k_m \cdots k_1)$, where

Notice $\lceil n/s \rceil = (k_m \cdots k_1)$, which has $m$ digits, and $n_1 = \lceil n/s \rceil$ could be larger than $\lceil n/s \rceil$ by one, which means $n_1$ either has $m$ digits or equal to $s^m$.

The case that $\lceil n/s \rceil$ has one more digit than $\lceil n/s \rceil$ could happen only one in the recursive procedure. So $n_m$ has either one digit or equal to $s$, in both case, we have $p(x) = y_{m+1,0}$.

iv) We obtain a $\sigma_s$ network realization of $p(x)$ by taking a concatenation of all the sub-networks in each iteration.

$$\Phi_{p_0}^2 = \Phi_a^{m+1} \circ \Phi_a^m \circ \cdots \circ \Phi_a^1.$$ \hfill (2.48)

Its number of nodes and nonzero weights are

$$N(\Phi_{p_0}^2) = \sum_{k=1}^{m+1} N(\Phi_a^k) = 2 + 2s + \sum_{i=2}^{m+1} (2 + n_i w) = \mathcal{O}\left(\frac{s^2 + 3s + 2}{3} \frac{s}{s-1} n\right).$$ \hfill (2.49)

$$M(\Phi_{p_0}^2) = 2 + 4s + \sum_{i=1}^{m} \left(2N(\Phi_a^{i+1}) + w \sum_{k=0}^{n_{i+1}} w\right) + N(\Phi_a^{m+1}) = \mathcal{O}\left(\frac{s^2 + 3s + 2}{9} \frac{s}{s-1} n\right).$$ \hfill (2.50)

The above construction produces a network with $m + 2$ layers which is optimal. But the numbers of nodes and nonzero weights are not optimal for large values of $s$. Next, we present an alternative construction method in following theorem that is optimal in both number of layers and number of nodes.

**Theorem 2.** If $p(x)$ is a polynomial of degree $s$ on $\mathbb{R}$, then it can be represented exactly by a $\sigma_s$ neural network with $[\log_s n] + 2$ layers, and number of nodes and non-zero weights are of order $\mathcal{O}(n)$ and $\mathcal{O}(sn)$, respectively.

**Proof.** 1) For polynomials of degree up to $s$, the formula (2.25) in Corollary 1.1 presents a one-hidden-layer network realization that satisfies the theorem.

2) Below, we give a realization with much less number of nodes and nonzero weights by adding one more hidden layer. We describe the new construction in following steps.

i) The first sub-network calculate $z_0 = x^s$ and $z_{0,1} = x$ using

$$\Phi_b^s = \Phi_a^1 \circ \Phi_b^1, \Phi_{dx} \in \Pi_{1,N_1,2}^1.$$ \hfill (2.51)

where the number of nodes in this sub-network is $N_1 = 2 + 2s$.

ii) In the second sub-network, we calculate

$$z_1 = z_0^s, \quad z_{1,j} = z_0^s, \quad j = 1, \ldots, s - 1; \quad y_{1,k} = \sum_{j=0}^{s-1} a_{k+j} z_{0,1}^{j}, \quad k = 0, \ldots, n_1 - 1,$

which can be implemented as

$$\begin{align*}
\Phi_b^s &= \Phi_{mo}^{s} \circ (\Phi_c \circ \Phi_a^1), \\
\Phi_c &= (\alpha_1, \beta_1), (\gamma_c, \Lambda_c), \\
\Phi_a^1 &= (\alpha_1, \beta_1, (A_a, b_a)), \\
\gamma_c &= (\gamma_1, \ldots, y_{1,s-1})^T, \\
\Lambda_a &= \lambda_3 (a_0, \ldots, \lambda s^2 a_{n_1-1})^T, \\
\alpha_1 &= \lambda_1 (\alpha_0, \ldots, \alpha_{s-1})^T, \\
b_a &= \lambda_1 (a_0, \ldots, c_1 (a_{n_1-1})^T).
\end{align*}$$ \hfill (2.52)

where $a_k = (a_{k-1}^s, a_k, a_{k+1}, a_{k+2}, \ldots, a_{k+s}^s)$. $\Phi_c$ is a network to realize $\{z_{1,j} \mid j = 1, \ldots, s - 1\}$, which is sketched in Fig. 2A. $\Phi_a^1$ is a network to realize $\{y_{1,k} \mid k = 0, \ldots, n_1 - 1\}$, which is sketched in Fig. 2B. Note that, according to Lemma 1 and Corollary 1.1, the number of nodes in $\Phi_a^1$ is $N_2 = 2 + 4s$. 

14
iii) For $i = 2, \ldots, m + 1$, the $(i + 1)$-th sub-network realize

$$z_i = z_{i-1}^i, \quad z_{i,j} = z_{i-1,j}^i, \quad j = 1, \ldots, s - 1; \quad y_{i,k} = \sum_{j=0}^{s-1} y_{i-1,k,s+j} z_{i-1,j}, \quad \text{for } k = 0, \ldots, n_i - 1,$$

which, according to Lemma 1 and Lemma 2 can be realized by a neural network $\Phi_b^i$ of only one hidden layer.

$$\Phi_b^i = \Phi_{m_0} \lor (\Phi_r) \lor \Phi_{2i}^i.$$

(2.53)

$\Phi_{2i}^i$ is a network to realize $\{y_{i,k} | k = 0, \ldots, n_i - 1\}$. It is composed of $n_i$ sub-nets $\Phi_{b,m,1}^i$ calculating multiplications. The number of nodes, and numbers of nonzero weights in $\Phi_b^i$ are

$$N(\Phi_b^i) = 2 + 2s + 4(s - 1)n_i; \quad (2.54)$$

$$M_i(\Phi_b^i) = 2 + 4s + 12(s - 1)n_i \quad M_2(\Phi_b^i) = 2 + 2(s - 1)s + 4s(s - 1)n_i.$$

(2.55)

At the end of the iteration, we have $p(x) = y_{m+1,0}$.

iv) The overall network is obtained by taking a concatenation of all the sub-networks in each iteration.

$$\Phi_{p_o} = \Phi_{b, m+1} \circ \Phi_{b, m} \circ \cdots \circ \Phi_{b, 0}.$$

(2.56)

This network has $m + 3$ layers. A straightforward calculation gives us

$$N(\Phi_{p_o}) = \sum_{k=1}^{m+1} N(\Phi_b^k) = (2 + 2s) + (2 + 4s) + \sum_{i=2}^{m+1} (2 + 2s + 4s(s - 1)n_i) = \Theta(4n),$$

(2.57)

$$M(\Phi_{p_o}) = (2 + 4s) + ((4 + 6s) + 4s^2 + 2s) + ((4 + 6s) + 4(s - 1)n_i) + \sum_{i=2}^{m} (4 + 6s + 4(s - 1)n_i + 2s + 4s(s - 1) + 1) + N(\Phi_{m+1}) = \Theta((8s + 16)n).$$

(2.58)

The proof is complete. The overall construction is summarized in Algorithm 4.

---

**Algorithm 2** PNet_Polynomial($n, s, a$)

**Input:** $n, s, a = (a_0, a_1, \ldots, a_n)$.

**Output:** $\Phi_{p_o}$ (with property $R_{s_i}(\Phi_{p_o})(x) = \sum_{k=0}^{n} a_k x^k$)

1: if $n \leq s$ then
2: Form $\Phi_{p_o} = \Phi_{p_o}^1$ given by (2.26).
3: else
4: Let $m = \lceil \log_2 n \rceil$
5: Form $\Phi_{p_0}^0$ given by (2.51)
6: Form $\Phi_{p_0}^1$ given by (2.52)
7: for $i = 2$ to $m + 1$ do
8: Form $\Phi_{p_0}^i$ given by (2.53)
9: Form $\Phi_{p_0} = \Phi_{p_o}^3$ given by (2.56)
10: return $\Phi_{p_o}$.
2.3. Error bounds of approximating univariate smooth functions

Now we analyze the error of approximating general smooth functions using RePU networks. Let $\Omega \subseteq \mathbb{R}^d$ be the domain on which the function to be approximated is defined. For the one dimensional case, we focus on $\Omega = [I := [-1, 1]]$. We denote the set of polynomials with degree up to $N$ defined on $\Omega$ by $P_N(\Omega)$, or simply $P_N$. Let $f_n^{\alpha, \beta}(x)$ be the Jacobi polynomial of degree $n$ for $n = 0, 1, \ldots$, which form a complete set of orthogonal bases in the weighted $L^2_{\omega^{\alpha, \beta}}(I)$ space with respect to weight $\omega^{\alpha, \beta} = (1-x)^\alpha (1+x)^\beta$, $\alpha, \beta > -1$. To describe functions with high order regularity, we define Jacobi-weighted Sobolev space $B_{m, \alpha, \beta}^m(I)$ as \cite{devore2001approximation}:

$$B_{m, \alpha, \beta}^m(I) := \left\{ u : \partial_x^k u \in L^2_{\omega^{\alpha, \beta}}(I), \ 0 \leq k \leq m \right\}, \quad m \in \mathbb{N}_0,$$

with norm

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

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

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

A detailed error estimate on the projection error $\pi_{\alpha, \beta}^m u - u$ is given in Theorem 3.35 of \cite{devore2001approximation}, by which we have the following theorem on the approximating error of general smooth functions using RePU networks.

**Theorem 3.** Let $\alpha, \beta > -1$. For any $u \in B_{m, \alpha, \beta}^m$, there exist a $\sigma_\alpha$ network $\Phi_{sN}^\epsilon$ with $L(\Phi_{sN}^\epsilon) = [\log(N) + 2, N(\Phi_{sN}^\epsilon) = \Theta(N), M(\Phi_{sN}^\epsilon) = \Theta(sN)]$, satisfying the following estimate

- If $0 \leq l \leq m \leq N + 1$, we have
  $$\| \partial_x^l \left( R_{\sigma_\alpha}(\Phi_{sN}^\epsilon) - u \right) \|_{\omega^{\alpha, \beta}} \leq c \sqrt{\frac{(N - m + 1)!}{(N - l + 1)!}} (N + m)^{(l-m)/2} \| \partial_x^m u \|_{\omega^{\alpha, \beta}},$$

- If $m > N + 1$, we have
  $$\| \partial_x^l \left( R_{\sigma_\alpha}(\Phi_{sN}^\epsilon) - u \right) \|_{\omega^{\alpha, \beta}} \leq c (2\pi N)^{-1/4} \left( \frac{\sqrt{\epsilon/2}}{N} \right)^{N-l+1} \| \partial_x^{N+1} u \|_{\omega^{\alpha, \beta}}.$$  

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

**Proof.** For any given $u \in B_{m, \alpha, \beta}^m$, there exists a polynomials $f = \pi_{\alpha, \beta}^m u \in P_N$. The projection error $\pi_{\alpha, \beta}^m u - u$ is estimated by Theorem 3.35 in \cite{devore2001approximation}, which is exactly $2.62$ and $2.63$ with $R_{\sigma_\alpha}(\Phi_{sN}^\epsilon)$ replaced by $\pi_{\alpha, \beta}^m u$. By Theorem 2 $f$ can be represented by a ReQU network (denoted by $\Phi_{sN}^\epsilon$) with no error, i.e. $R_{\sigma_\alpha}(\Phi_{sN}^\epsilon) = \pi_{\alpha, \beta}^m u$. We thus obtain estimate $2.62$ and $2.63$. \hfill $\square$

**Remark 5.** Note that when $N \gg m$, the $L^2$ convergence rate given by $2.62$ is of order $\Theta(N^{-m})$, which by the optimal nonlinear approximation theory developed by DeVore, Howard and Micchelli \cite{devore1985optimal}, is optimal if the network parameters depend continuously on the approximated function.

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

**Theorem 4.** For any given function $f(x) \in B_{m, 0}^m$ with norm less than 1, where $m$ is either a fixed positive integer or infinity, there exists a RePU network $\Phi_{sN}^\epsilon$ can approximate $f$ within an error tolerance $\epsilon$, i.e.

$$\| R_{\sigma_\alpha}(\Phi_{sN}^\epsilon) - f \|_{L^2(I)} \leq \epsilon.$$ 

The number of layers $L$, numbers of nodes $N$ and nonzero weights $M$ can be bounded as

\[16\]
• If \( m \) is a fixed positive integer, then \( L = \Theta\left(\frac{1}{m} \log \frac{1}{\varepsilon}\right), N = \Theta\left(\varepsilon^{-1} \ln \varepsilon\right) \) and \( M = \Theta\left(s \varepsilon^{-\frac{1}{m}}\right)\).

• If \( m = \infty \), then \( L = \Theta\left(\log \left(\frac{1}{\varepsilon}\right)\right), N = \Theta\left(\varepsilon^{-1} \ln \varepsilon\right) \), and \( M = \Theta\left(s \varepsilon^{-\frac{1}{m}}\right)\). Here \( \gamma_0 = \ln \left(\frac{1}{\varepsilon}\right)\).

Proof. For a fixed \( m \), or \( N \gg m \), we obtain from (2.62) that

\[
\|R_{\varepsilon}(\Phi_N^m) - u\|_{\ell^2} \leq cN^{-m}\|\bar{a}_N\|_{\ell^{m, m, \beta}}. \tag{3.65}
\]

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 \( N = \left(\frac{\varepsilon}{\varepsilon^2}\right)^{1/m} \). For fixed \( m \), we have \( N = \Theta\left(\varepsilon^{-1} \ln \varepsilon\right) \), the depth of the corresponding RePU network is \( L = \Theta\left(\frac{1}{m} \log \frac{1}{\varepsilon}\right) \), and the number of nonzero weights is \( M = \Theta\left(s \varepsilon^{-\frac{1}{m}}\right)\).

For \( m = \infty \), from equation (2.63), we have

\[
\|R_{\varepsilon}(\Phi_N^m) - u\|_{\ell^2} \leq c\left(\frac{\sqrt{e/2}}{N}\right)^{N+1} \|u\|_{\tilde{B}_{\alpha, \beta}}^\infty \|u\|_{\tilde{B}_{\alpha, \beta}}. \tag{3.66}
\]

where \( c' \) is a positive constant, and \( \gamma = \Theta\left(\ln N - \frac{1}{3}\right) \) can be larger than any fixed positive number for sufficient large \( N \). To approximate a function with \( B^m_{\alpha, \beta}(I) \) norm less than 1 with error \( \varepsilon = c' \varepsilon^{-N} \), one needs to take \( N = \frac{1}{\varepsilon} \ln \left|\left(\frac{e}{e'\varepsilon}\right)\right| \) for \( N > e^{1.5} \), from which we get \( \gamma = \Theta\left(\ln N\right) = \Theta\left(\ln \left(\frac{e}{e'\varepsilon}\right)\right) \), thus \( N = \Theta\left(\frac{1}{\varepsilon} \ln \left(\frac{e}{e'\varepsilon}\right)\right) \). The depth of the corresponding RePU network is \( L = \Theta\left(\log \left(\frac{1}{\varepsilon}\right)\right) \). The number of nonzero weights is \( \Theta\left(s \varepsilon^{-\frac{1}{m}}\right)\). \( \square \)

3. Approximation of multivariate smooth functions

In this section, we discuss the approximation of multivariate smooth functions by RePU networks. Similar to the univariate case, we first study the representation of polynomials then discuss the results for general smooth functions.

3.1. Approximating multivariate polynomials

**Theorem 5.** If \( f(x) \) is a multivariate polynomial with total degree \( n \) on \( \mathbb{R}^d \), then there exists a \( \sigma_s \) neural network \( \Phi^{d}_{\text{mpo}} \) having \( d[\log n] + 1 \) hidden layers with no more than \( \Theta\left(C_n + d\right) \) activation functions and \( \Theta\left(s C_n + d\right) \) non-zero 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 \( n \geq s + 1 \) (The cases \( n \leq s \) are similar but easier, so we omit here). To represent \( f(x, y) \) exactly with a \( \sigma_s \) neural network basing the results on 1-dimensional case given in Theorem 2 we first rewrite \( f(x, y) \) as

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

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

(1) Since \( \Phi^y \) takes \( a_i^y, i = 0, \ldots, n \) 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 \( \Phi_i^y, i = 0, \ldots, n-1 \) is \( [\log n] + 2 \), so we can lift \( x \) to layer \( [\log n] + 2 \) using a concatenation of multiple \( id_X(\cdot) \) operations. Similarly, we also keep a record of input \( y \) in each layer using multiple \( id_Y(\cdot) \), such that \( \Phi_i^y, i = 1, \ldots, n-1 \) can start from appropriate layer and generate output exactly at layer \( [\log n] + 2 \). The overall cost for recording \( x, y \) in layers \( 1, \ldots, [\log n] + 2 \) is about \( 4s([\log n] + 2) \), which is negligible comparing to the overall cost.
(2) While realizing $\sum_{i=0}^{n} a_i^T x^i$, the coefficients $a_i^T, i = 0, \ldots, n$ 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 and second layer of the network. i.e. using approach for $y_{1,k}, i \geq 2$ in (2.53) for $y_{1,k}$ as well. This will increase the nodes in this layer from $\mathcal{O}(n)$ to $\mathcal{O}(sn)$, but since $n > s$, this does not change the overall scaling of the total number of nodes.

By a direct calculation, we find the number of layers, number of nodes and nonzero weights in this realization can be bounded by $2[\log_s n] + 2, \mathcal{O}(C_{n+2}^m)$, and $\mathcal{O}(sC_{n+2})$.

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

Using similar approach as in Theorem 5, one can easily prove the following theorem.

Theorem 6. 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 $\sigma_s$ network having $d[\log_s N] + 1$ hidden layers with no more than $\mathcal{O}(N^d)$ activation functions and $\mathcal{O}(sN^d)$ non-zero weights, can represent $f_N$ with no error.

3.2. Error bound of approximations of multivariate smooth functions

For a vector $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, we define $|x|_1 := |x_1| + \cdots + |x_d|$, $|x|_\infty := \max_{i=1}^d |x_i|$. Define high dimensional Jacobi weight $\omega_{\alpha, \beta} := \omega^{\alpha_1, \beta_1} \cdots \omega^{\alpha_d, \beta_d}$. We define multidimensional Jacobi-weighted Sobolev space $B^m_{\alpha, \beta}(I^d)$ as (26):

$$B^m_{\alpha, \beta}(I^d) := \left\{ u(x) \mid \partial_{x_i}^j u := \partial_{x_{i_1}}^{\beta_1} \cdots \partial_{x_{i_d}}^{\beta_d} u \in L_{|| \omega_{\alpha, \beta} | |_{(k)}}, \quad k \in \mathbb{N}_0^d, \quad |k|_1 \leq m \right\}, \quad m \in \mathbb{N}_0.$$  (3.2)

with norm and semi-norm

$$\| u \|_{B^m_{\alpha, \beta}} := \left( \sum_{|k|_1 \leq m} \left\| \partial_{x_i}^j u \right\|_{L_{\omega_{\alpha, \beta} | |_{(k)}}}^2 \right)^{1/2}, \quad \| u \|_{B^m_{\alpha, \beta}} := \left( \sum_{|k|_1 \leq m} \left\| \partial_{x_i}^j u \right\|_{L_{\omega_{\alpha, \beta} | |_{(k)}}}^2 \right)^{1/2}.$$  (3.3)

Define the $L^2_{\omega_{\alpha, \beta}}$-orthogonal projection $P^m_{N_{\alpha, \beta}} : L^2_{\omega_{\alpha, \beta}}(I^d) \rightarrow Q_N^d(I^d)$ as

$$\left( P^m_{N_{\alpha, \beta}} u - u, v \right)_{\omega_{\alpha, \beta}} = 0, \quad \forall v \in P^d_{N_{\alpha, \beta}}(I^d).$$

Then for $u \in B^m_{\alpha, \beta}$, we have the following error estimate (see e.g. (26))

$$\| P^m_{N_{\alpha, \beta}} u - u \|_{L^2_{\omega_{\alpha, \beta}}(I^d)} \leq c N^{-m} \| u \|_{B^m_{\alpha, \beta}}, \quad 1 \leq m \leq N,$$  (3.4)

where $c$ is a general constant. Combining (3.4) and Theorem 6 we reach to the following upper bound for the $\varepsilon$-approximation of functions in $B^m_{\alpha, \beta}(I^d)$ space.

Theorem 7. For any $u \in B^m_{\alpha, \beta}(I^d)$, with $\| u \|_{B^m_{\alpha, \beta}(I^d)} \leq 1$, there exists a $\sigma_s$ neural network $\Phi_\varepsilon$ having $\mathcal{O} \left( \frac{d}{m} \log_s \frac{1}{\varepsilon} + d \right)$ hidden layers with no more than $\mathcal{O}(\varepsilon^{-d/m})$ nodes of $\sigma_s$-neural network with complexity $\mathcal{O}(s\varepsilon^{-d/m})$ non-zero weights, approximate $u$ with $L^2_{\omega_{\alpha, \beta}}(I^d)$ error less than $\varepsilon$, i.e.

$$\| R_{\sigma_s}(\Phi_\varepsilon u) - u \|_{L^2_{\omega_{\alpha, \beta}}(I^d)} \leq \varepsilon. $$  (3.5)

3.3. 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 RePU neural network with complexity $\mathcal{O}(\varepsilon^{-d/m})$. When $m$ is much smaller than $d$, we see the network complexity has an exponential dependence on $d$. However, in a lot of applications, high-dimensional problem may have low intrinsic dimension (28), for those applications, we may first do a dimension reduction, then use the $\sigma_s$ neural network construction proposed above to approximate the reduced problem. On the other hand, for high-dimensional functions with bounded mixed derivatives, we can use sparse grid or hyperbolic cross approximation to lessen the curse of dimensionality.
3.3.1. A brief review on hyperbolic cross approximations

We introduce hyperbolic cross approximation by considering a tensor product function: \( f(x) = f_1(x_1) f_2(x_2) \cdots f_d(x_d) \). 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

\[
f_i(x) = \sum_{k=0}^{\infty} b_k^{(i)} \phi_k(x), \quad |b_k^{(i)}| \leq c \tilde{k}^{-r}, \quad i = 1, 2, \ldots, d,
\]

(3.6)

where \( c \) and \( r \geq 1 \) are constants depending on the regularity of \( f_i, \tilde{k} := \max(1, k) \). So we have an expansion for \( f \) as

\[
f(x) = \prod_{i=1}^{d} \left( \sum_{k \in \mathbb{N}_0^d} b_k^{(i)} \phi_k(x_i) \right) = \sum_{k \in \mathbb{N}_0^d} b_k \phi_k(x), \quad \text{where } |b_k| = |b_k^{(1)}| \cdots |b_k^{(d)}| \leq c \tilde{k}^{-r}.
\]

(3.7)

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

\[
f_N := \sum_{k \in I_N^d} b_k \phi_k(x),
\]

(3.8)

where

\[
I_N^d := \left\{ k = (k_1, \ldots, k_d) \in \mathbb{N}_0^d \mid \tilde{k} \leq N \right\}
\]

(3.9)

is the hyperbolic cross index set. We call \( f_N \) defined by (3.8) a hyperbolic cross approximation of \( f \).

For general functions defined on \( I^d \), we choose \( \phi_k \) to be multivariate Jacobi polynomials \( J_n^\alpha Beta \), and define the hyperbolic cross polynomial space as

\[
X_N^d := \text{span}\{ J_n^\alpha Beta \mid n \in \mathbb{N}_0^d \}.
\]

(3.10)

Note that the definition of \( X_N^d \) doesn't depend on \( \alpha \) and \( \beta \). \( \{ J_n^\alpha Beta \} \) 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

\[
\mathcal{K}_n^\alpha Beta(I^d) := \left\{ u(x) : \partial_x^k u \in L_{\omega_n^\alpha Beta}^2(I^d), 0 \leq |k|_{\infty} \leq m \right\}, \quad \text{for } m \in \mathbb{N}_0,
\]

(3.11)

with norm and semi-norm

\[
\| u \|_{\mathcal{K}_n^\alpha Beta} := \left( \sum_{|k|_{\infty} \leq m} \| \partial_x^k u \|_{L_{\omega_n^\alpha Beta}^2}^2 \right)^{1/2}, \quad |u|_{\mathcal{K}_n^\alpha Beta} := \left( \sum_{|k|_{\infty} = m} \| \partial_x^k u \|_{L_{\omega_n^\alpha Beta}^2}^2 \right)^{1/2}.
\]

(3.12)

For any \( u \in \mathcal{K}_n^\alpha Beta = B_n^\alpha Beta \), the hyperbolic cross approximation can be defined as a projection as

\[
\pi_n^\alpha Beta (\cdot, u) \phi_{\alpha Beta} = 0, \quad \forall \phi_{\alpha Beta} \in X_N^d.
\]

(3.13)

Then we have the following error estimate about the hyperbolic cross approximation [39]:

\[
\| \partial_x^l \pi_n^\alpha Beta (\cdot, u) \phi_{\alpha Beta} \alpha Beta \leq D_n |l|_{\infty} - m \| u \|_{X_N^d}^\alpha Beta, \quad 0 \leq l \leq m, \quad m \geq 1,
\]

(3.14)

where \( D_n \) is a constant independent of \( N \). It is known that the cardinality of \( \chi_N^d \) is of order \( O(N(\log N)^{d-1}) \). The above error estimate says that to approximation a function \( u \) with \( |u|_{X_N^d}^{\alpha Beta} \leq 1/D_n \) with an error tolerance \( \epsilon \), one need no more than \( \Theta \left( \epsilon^{-1/m} (\frac{1}{m} \log \frac{1}{\epsilon})^{d-1} \right) \) polynomials, the exponential dependence on \( d \) is weakened.

In practice, the exact hyperbolic cross projection is not easy to calculate. An alternate approach is the sparse grids [40][41], which use hierarchical interpolation schemes to build an hyperbolic cross like approximation of high dimensional functions [42][43].
3.3.2. Error bounds of approximating some high-dimensional smooth functions

Now we discuss the RePU network approximation of high-dimensional smooth functions. Our approach bases on high-dimensional hyperbolic cross polynomial approximations. 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: There exists a set of bases composed of only monomials belonging to $P_C$, and for any term $p(x)$ in this basis set, all of its derivatives $\partial^a_k p(x)$, $k \in N^d$ belongs to $P_C$. It is easy to verify that both the hyperbolic cross polynomial space $X^d$ and sparse grid polynomial interpolation space $V^d$ (see \cite{33,34}) are complete. For a complete polynomial space, we have the following RePU network representation results.

**Theorem 8.** 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 $\sigma_s$ neural network having no more than $\sum_{i=1}^d [\log_i N_i] + 1$ hidden layers, no more than $\Theta(n)$ activation functions and $\Theta(n)$ non-zero weights, can represent $f$ with no error. Here $N_i$ is the maximum polynomial degree in $i$-th dimension in $P_C$.

**Proof.** The proof is similar to Theorem 5. First, $f$ can be written as linear combinations of monomials.

$$f(x) = \sum_{k \in C} a_k x_k,$$

(3.15)

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

$$f(x) = \sum_{k_d=0}^{N_d} a_{k_d} x_{d-1} d \cdot \sum_{(k_1, \ldots, k_{d-1}) \in C} a_{k_1, \ldots, k_{d-1}} x_{d-1},$$

(3.16)

where $x_{d-1}$ are $d-1$ dimensional complete index sets that depend on the index $k_d$. If each term in $a_{k_d} x_{d-1} d \cdot$, $k_d = 0, 1, \ldots, N_d$ can be exactly represented by a $\sigma_s$ network with no more than $\sum_{i=1}^d [\log_i N_i] + 1$ hidden layers, no more than $\Theta(\log(N/d))$ nodes and $\Theta((\log(N/d))$ non-zero weights, then $f(x)$ can be exactly represented by a $\sigma_s$ neural network with no more than $\sum_{i=1}^d [\log_i N_i] + 1$ hidden layers, no more than $\Theta(n)$ nodes and non-zero weights. So, by mathematical induction, we only need to prove that when $d = 1$ the theorem is satisfied, which is true by Theorem 5. \hfill $\Box$

**Remark 6.** According to Theorem 5, we have that: For any $f \in X^d$, there is a RePU network having no more than $d [\log N] + 1$ hidden layers, no more than $\Theta(N(d-1))$ activation functions and $\Theta(N(d-1))$ non-zero weights, can represent $f$ with no error. Combine the results with (3.14) and we can obtain the following theorem.

**Theorem 9.** For any function $u \in \mathcal{X}_m^a, m \geq 1$ with $|u|_{\mathcal{X}_m^a} \leq 1/D_1$, any $\varepsilon \geq 0$, there exists a RePU network $\Phi_{\varepsilon}^u$ with no more than $d(1 \log_{\varepsilon} 1/\varepsilon) + 2$ layers, no more than $\Theta(\varepsilon^{-m}(1 \log_{\varepsilon} 1/\varepsilon))$ nodes and $\Theta(\varepsilon^{-m}(1 \log_{\varepsilon} 1/\varepsilon))$ non-zero weights, such that

$$\|R_{\varepsilon}(\Phi_{\varepsilon}^u) - u\|_{\omega_a, \varepsilon} \leq \varepsilon.$$

(3.17)

**Remark 7.** Here, we bound the weighted $L^2$ approximation error by using the corresponding hyperbolic cross spectral projection error estimation developed in \cite{29}. However, high-dimensional hyperbolic cross spectral projection is hard to calculate. In practice, we use efficient sparse grid spectral transforms developed in \cite{33} and \cite{34} to approximate the projection. After a numerical network is built, one may further train it to obtain a network function that is more accurate than the sparse grid interpolation. Note that the fast sparse transform can be extended to tensor-product unbounded domain using the mapping method \cite{35}.

4. Summary

In this paper, deep neural network realizations of univariate polynomials and multivariate polynomials using general RePU as activation functions are proposed with detailed constructive algorithms. The constructed RePU neural networks have optimal number of hidden layers and optimal number of activation nodes. By using
this construction, we also prove some optimal upper error bounds of approximating smooth functions in Sobolev space using RePU networks. The optimality is indicated by the optimal nonlinear approximation theory developed by DeVore, Howard and Micchelli for the case that the network parameters depend continuously on the approximated function. The constructive proofs reveal clearly the close connection between the spectral method and deep RePU network approximation.

Even though we did not apply the proposed RePU networks to any real applications in this paper, the good properties of the proposed networks suggest that they have potential advantages over other types of networks in approximating functions with good smoothness. In particular, it suits situations where the loss function contains some derivatives of the network function, in such a case, deep ReLU networks are known hard to use with usual training methods.

Appendix

The appendix section is devoted to proof Lemma 3. We first present the following lemma which can be proved by induction.

**Lemma 3.** For \( s \in \mathbb{N} \) we have

\[
(2^s - 1) \prod_{k=1}^{s} x_k = \left( \sum_{k=1}^{s} x_k \right)^s + \sum_{1 < i_1 < \ldots < i_k} (-1)^k S_{i_1, \ldots, i_k},
\]

where

\[
S_{i_1, \ldots, i_k} = (x_1 + \cdots + (-1)x_{i_1} + \cdots + (-1)x_{i_k} + \cdots + x_s)^s.
\]

**Corollary 3.1.** For \( s \in \mathbb{N} \) and \( n_1 + n_2 = t \in \{0, 1, \ldots, s\} \), \( n_1, n_2 \in \{0, 1, \ldots, t\} \), we have

\[
(2^s - 1)! x^{n_1} y^{n_2} = \left[ n_1 x + n_2 y + (s - t) \right]^s + \sum_{k=1}^{s-t} (-1)^k C_{s-t}^k \left[ n_1 x + n_2 y + (s - t - 2k) \right]^s
\]

\[
+ \sum_{k=1}^{s-t} (-1)^k \sum_{r=\max(k, s-t-k)}^{\min(s, r-1)} C_{s-t}^k C_{n_1-1}^{r-k} C_{n_2}^{s-r} S_{j, r, k},
\]

where

\[
S_{j, r, k} := \left[ (n_1 - 2j)x + (n_2 - 2(r - j))y + (s - (n_1 + n_2) - 2(k - r)) \right]^s.
\]

**Proof.** We take \( x_1 = \cdots = x_{n_1} = x, x_{n_1+1} = \cdots = x_t = y, x_{t+1} = \cdots = x_s = 1 \) in Lemma 3. Denote

\[
A_t = \{x_1, x_2, \ldots, x_t\}, \quad B_k = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\},
\]

and let \( \#(A_t \cap B_k) \) be the number of elements in both \( A_t \) and \( B_k \). Then the second term on the left hand side of (4.1) can be summed in two groups:

- The first group include the cases that no term in \( B_k \) is included in \( A_t \), so we get \( x_{i_1} = \cdots = x_{i_k} = 1 \). Each \( S_{i_1, \ldots, i_k} \) term in this case is equal to \( [n_1 x + n_2 y + (s - t - 2k)]^s \), there are \( C_{s-t}^k \) such terms.

- The second group includes the cases that there exist at least one term in \( B_k \) is contained in \( A_t \). We let \( r = \#(A_t \cap B_k) > 0, j = \#(A_{n_1} \cap B_k) \), then we have

\[
\max(1, k - (s - t)) \leq r \leq \min(t - 1, k),
\]

\[
\max(0, r - n_2) \leq j \leq \min(r, n_1 - 1).
\]

Each \( S_{i_1, \ldots, i_k} \) term in this case is equivalent to (4.3). There are in total \( C_{s-t}^k C_{n_1-1}^{r-k} C_{n_2}^{s-r} \) such terms.
where terms, that is (4.5) can be written as
\[ z \]

Denote by \( R \)

First, by taking \( n_1 = 1, n_2 = n = t - 1 \) in Corollary 3.1 and exchange the positions of \( x, y \), we get
\[ (2^{s-1} s)! x^n = \left[ n x + y + (s - (n + 1)) \right] ^s + \sum _{k=1} ^{s-(n+1)} (-1)^k \frac{C_s}{C_k} \left[ n x + y + (s - 2(k - r)) \right] ^s \]
\[ = \sum _{k=1} ^{s-(n+1)} (-1)^k \frac{C_s}{C_k} \left[ n x + y + (s - (n + 1) - 2) \right] ^s \]
\[ + \sum _{j=0} ^{s-(n+1)} \sum _{r=0} ^n (-1)^{j+r} c^j_s \left[ n x + y + (s - (n + 1) - 2j) \right] ^s \]

From above derivation, we see that \( x^n y \) can be represented as a linear combination of \( (n + 1) \times (s - n) \) \( \rho_s(\cdot) \) terms, that is
\[ x^n y = \sum _{j=0} ^{s-(n+1)} \sum _{r=0} ^n \gamma ^j_s (n x + y + (s - (n + 1) - 2j)) \]
(4.5)

where
\[ \gamma ^j_s = \frac{(-1)^{j+r}}{(2^{s-1} s)!} \frac{C_s}{C_k} c^j_s \]
(4.6)

Denote by \( z_{k+1} := (k, k-2, \ldots, -k) ^T \in \mathbb{R} ^{k+1} \), and \( 1_k := (1, 1, \ldots, 1) ^T \in \mathbb{R} ^k \), for \( k \in \mathbb{Z} \). For a matrix \( A = (a_k) \in \mathbb{R} ^{m \times n} \), define its vectorization \( \text{vec}(A) := (a_1, \ldots, a_m, 1, \ldots, a_{m-n}, \ldots, a_{m-n}) ^T \). For two vectors \( a \in \mathbb{R} ^m, b \in \mathbb{R} ^n \), define \( a \otimes b := (a_i b_j) _{i=1} ^{m} _{j=1} ^{n} \in \mathbb{R} ^{m \times n} \). Denote \( \Gamma _{s,n} = (\gamma ^j_s) _{j=0} ^{s-(n+1)} \in \mathbb{R} ^{(s-n)(n+1)} \). Using these definitions and notations, (4.5) can be written as
\[ x^n y = \gamma ^T _{s,n} \sigma (a z_{2,1} x + a z_{2,2} y + b z_{2,1}) \]
(4.7)

where
\[ \gamma _{2,n} = \text{vec}(\Gamma _{s,n}), \quad a z_{2,1} = \text{vec}\left(a_0 \otimes \text{vec}(1_{s-n} \otimes z_{n+1})\right), \quad a z_{2,2} = \text{vec}\left(a_0 \otimes \text{vec}(z_{s-n} \otimes 1_{n+1})\right) \]
(4.8)

The length of those coefficients are all \( 2(s-n)(n+1) \). The lemma is proved.

\[ \square \]

Acknowledgments

The last author is indebted to Prof. Jie Shen and Prof. Li-Lian Wang for their stimulating conversations on spectral methods. The authors would like to thank Prof. Christoph Schwab and Prof. Hrushikesh N. Mhaskar for providing us some related references. This work was partially supported by China National Program on Key Basic Research Project 2015CB856003, NNSFC Grant 11771439, 91852116, and China Science Challenge Project, no. TZ2018001.
