A comparison of deep networks with ReLU activation function and linear spline-type methods

Konstantin Eckle, Johannes Schmidt-Hieber *

Leiden University, Mathematical Institute, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

Article Info

Article history:
13 April 2018 Received
16 November 2018 Revised
20 November 2018 Accepted
Available online 4 December 2018

Keywords:
Deep neural networks
Nonparametric regression
Splines
MARS
Faber-Schauder system
Rates of convergence

Abstract

Deep neural networks (DNNs) generate much richer function spaces than shallow networks. Since the function spaces induced by shallow networks have several approximation theoretic drawbacks, this explains, however, not necessarily the success of deep networks. In this article we take another route by comparing the expressive power of DNNs with ReLU activation function to linear spline methods. We show that MARS (multivariate adaptive regression splines) is improper learnable by DNNs in the sense that for any given function that can be expressed as a function in MARS with $M$ parameters there exists a multilayer neural network with $O(M \log(M/\epsilon))$ parameters that approximates this function up to sup-norm error $\epsilon$. We show a similar result for expansions with respect to the Faber–Schauder system. Based on this, we derive risk comparison inequalities that bound the statistical risk of fitting a neural network by the statistical risk of spline-based methods. This shows that deep networks perform better or only slightly worse than the considered spline methods. We provide a constructive proof for the function approximations.

© 2018 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Training of DNNs is now state of the art for many different complex tasks including detection of objects on images, speech recognition and game playing. For these modern applications, the ReLU activation function is standard. One of the distinctive features of a multilayer neural network with ReLU activation function (or ReLU network) is that the output is always a piecewise linear function of the input. But there are also other well-known nonparametric estimation techniques that are based on function classes built from piecewise linear functions. This raises the question in which sense these methods are comparable.

We consider methods that consist of a function space $\mathcal{F}$ and an algorithm/learner/estimator that selects a function in $\mathcal{F}$ given a dataset. In the case of ReLU networks, $\mathcal{F}$ is given by all network functions with fixed network architecture and the chosen algorithm is typically a version of stochastic gradient descent. A method can perform badly if the function class $\mathcal{F}$ is not rich enough or if the algorithm selects a bad element in the class. The class of candidate functions $\mathcal{F}$ is often referred to as expressiveness of a method.

Deep networks have a much higher expressiveness than shallow networks, cf. Eldan and Shamir (2016), Liang and Srikant (2016), Mhaskar (1993), Poggio, Mhaskar, Rosasco, Miranda, and Liao (2017), Telgarsky (2016), Yarotsky (2017). Shallow ReLU networks have, however, several known drawbacks. It requires many nodes/units to localize and to approximate for instance a function supported on a smaller hypercube, cf. Chui, Li, and Mhaskar (1996). A large number of parameters is moreover necessary to approximately multiply inputs. While the literature has mainly focused on the comparison deep versus shallow networks, it is necessary to compare deep networks also with methods that have a similar structure but do not suffer from the shortcomings of shallow networks.

There is a vast amount of literature on spline based approximation, cf. Chui (1988), Chui and Jiang (2013), de Boor, Höllig, and Riemenschneider (1993), Micchelli (1995). For a direct comparison with ReLU networks it is natural to focus on spaces spanned by a piecewise linear function system. In this article we compare deep ReLU networks to well-known spline methods based on classes of piecewise linear functions. More specifically, we consider multivariate adaptive regression splines (MARS) (Friedman, 1991) and series expansion with respect to the Faber–Schauder system (Faber, 1910; van der Meulen, Schauer, & van Waaik, 2017).

MARS is a widely applied method in statistical learning (Hastie, Tibshirani, & Friedman, 2009). Formally, MARS refers to the algorithm describing the selection of a function from a specific
class of piecewise linear candidate functions called the MARS function class. Similarly as multilayer neural networks, the MARS function class is built from a system of simple functions and has width and depth like parameters. In contrast to DNNs, MARS also uses tensorization to generate multivariate functions. The Faber–Schauder functions are the integrals of the Haar wavelet families and are hence piecewise linear. Series expansion with respect to the Faber–Schauder system results therefore in a piecewise linear approximation. Both MARS and Faber–Schauder class satisfy an universal approximation theorem, see Section 2.2 for a discussion.

We show that for any given function that can be expressed as a function in the MARS function class or the Faber–Schauder class with M parameters there exists a multilayer neural network with \(O(M \log(M/\varepsilon))\) parameters that approximates this function up to sup-norm error \(\varepsilon\). We also show that the opposite is false. There are functions that can be approximated up to error \(\varepsilon\) with \(O(\log(1/\varepsilon))\) network parameters but require at least of order \(\varepsilon^{-1/2}\) many parameters if approximated by MARS or the Faber–Schauder class.

The result provides a simple route to establish approximation properties of deep networks. For the Faber–Schauder class, approximation rates can often be deduced in a straightforward way and these rates then carry over immediately to approximations by deep networks. As another application, we prove that for nonparametric regression, fitting a DNN will never have a much larger risk than MARS or series expansion with respect to the Faber–Schauder system. Numerical simulations can be found in Section 2016.

\[\text{Definition 2.1 (MARS Function Class).} \] For any positive integer \(M\) and any constants \(F, C > 0\), define the family of functions

\[\mathcal{F}_{\text{MARS}}(M, C) = \left\{ f = \beta_0 + \sum_{m=1}^M \beta_m h_m : h_m \text{ is of the form (1)}, \right\} \]

\[|\beta_m| \leq C, \|f\|_{\infty} \leq F.\]

All functions are defined on \([0, 1]^d\). For notational convenience, the dependence on the constant \(F\) is omitted.

For regression, the least squares loss is the natural choice. Least-squares minimization over the whole function space is, however, computationally intractable. Instead, MARS – proposed by Friedman (1991) – is a greedy algorithm that searches for a set of basis functions \(h_i\) that explain the data well. Starting with the constant function, in each step of the procedure two basis functions are added. The new basis functions are products of an existing basis function with a new factor \((x_i - t_i)\), where \(t_i\) coincides with the \(j\)th component of one of the design vectors. Among all possible choices, the basis functions that lead to the best least-squares fit are selected. The algorithm is stopped after a prescribed number of iterations. It can be appended by a backwards deletion procedure that iteratively deletes a fixed number of selected basis functions with small predictive power. Algorithm 1 provides additional details.

**Input**: Integer \(M\) and sample \((X_i, Y_i), i = 1, \ldots, n\)  
**Output**: \(2M + 1\) basis functions of the form (1)

For \(k\) from 1 to \(M'\) do

\[
\begin{align*}
\text{for } g & \in S \text{ do} \\
|f(g)| \leftarrow & \{ f : x \mapsto g(x_1, \ldots, x_d) \text{ is constant}\} \\
\text{end}
\end{align*}
\]

\[
(g^*, f^*, t^*_p) \leftarrow \text{argmin over } g \in S, j \in f(g), t_p \in \left\{X_{ij}, i = 1, \ldots, n\right\} \text{ of the least squares fits for the models}
\]

\[
S \leftarrow S \cup \{ x \mapsto (x_i - t^*_p)g^*(x), x \mapsto (t^*_p - x_i)g^*(x) \}.
\]

end

return \(S\)

**Algorithm 1**: MARS forward selection

In the original article Friedman (1991) the values for \(t_i\) are restricted to a smaller set. The least squares functional can be replaced by any other loss. The runtime of the algorithm is of the order \((M')^2\), where the key step is the computation of the argmin which is easily parallelizable. Friedman (1993) considers a variation of the MARS procedure where the runtime dependence on \(M\) is only quadratic.

2.1 MARS – Multivariate Adaptive Regression Splines

Consider functions of the form

\[h_{ij}(x_1, \ldots, x_d) = \prod_{j=1}^d \{ t_j(x_j - t_j) \}, \quad \mathbf{x} = (x_1, \ldots, x_d)^T \in [0, 1]^d, \]

with \(1 \leq l \leq \cdots \leq d\) an index set, \(s_j \in \{-1, 1\}\) and \(t = (t_j)_{j \in l} \in [0, 1]^{|l|}\) a shift vector. These are piecewise linear functions in each argument whose supports are hyperrectangles. Suppose we want to find a linear combination of functions \(h_{ij}\) that best explains the data. To this end, we would ideally minimize a given loss over all possible linear combinations of at most \(M\) of these functions, where \(M\) is a predetermined cut-off parameter. Differently speaking, we search for an element in the following function class.

**Definition 2.1 (MARS Function Class).** For any positive integer \(M\) and any constants \(F, C > 0\), define the family of functions

\[\mathcal{F}_{\text{MARS}}(M, C) = \left\{ f = \beta_0 + \sum_{m=1}^M \beta_m h_m : h_m \text{ is of the form (1)}, \right\} \]

\[|\beta_m| \leq C, \|f\|_{\infty} \leq F.\]

All functions are defined on \([0, 1]^d\). For notational convenience, the dependence on the constant \(F\) is omitted.

For regression, the least squares loss is the natural choice. Least-squares minimization over the whole function space is, however, computationally intractable. Instead, MARS – proposed by Friedman (1991) – is a greedy algorithm that searches for a set of basis functions \(h_i\) that explain the data well. Starting with the constant function, in each step of the procedure two basis functions are added. The new basis functions are products of an existing basis function with a new factor \((x_i - t_i)\), where \(t_i\) coincides with the \(j\)th component of one of the design vectors. Among all possible choices, the basis functions that lead to the best least-squares fit are selected. The algorithm is stopped after a prescribed number of iterations. It can be appended by a backwards deletion procedure that iteratively deletes a fixed number of selected basis functions with small predictive power. Algorithm 1 provides additional details.

**Input**: Integer \(M\) and sample \((X_i, Y_i), i = 1, \ldots, n\)  
**Output**: \(2M + 1\) basis functions of the form (1)

For \(k\) from 1 to \(M'\) do

\[
\begin{align*}
\text{for } g & \in S \text{ do} \\
|f(g)| \leftarrow & \{ f : x \mapsto g(x_1, \ldots, x_d) \text{ is constant}\} \\
\text{end}
\end{align*}
\]

\[
(g^*, f^*, t^*_p) \leftarrow \text{argmin over } g \in S, j \in f(g), t_p \in \left\{X_{ij}, i = 1, \ldots, n\right\} \text{ of the least squares fits for the models}
\]

\[
S \leftarrow S \cup \{ x \mapsto (x_i - t^*_p)g^*(x), x \mapsto (t^*_p - x_i)g^*(x) \}.
\]

end

return \(S\)

**Algorithm 1**: MARS forward selection

In the original article Friedman (1991) the values for \(t_i\) are restricted to a smaller set. The least squares functional can be replaced by any other loss. The runtime of the algorithm is of the order \((M')^2\), where the key step is the computation of the argmin which is easily parallelizable. Friedman (1993) considers a variation of the MARS procedure where the runtime dependence on \(M\) is only quadratic.
There are many higher-order generalizations of the MARS procedure. E.g., Friedman (1991) proposed to use truncated power basis functions

\[
(s_j(x_j - t_j))^q, \quad q \in \mathbb{N},
\]

to represent \(q\)th order splines. Another common generalization of the MARS algorithm allows for piecewise polynomial basis functions in each argument of the form

\[
h_{K,t}(x_1, \ldots, x_d) = \prod_{j=1}^{K} (s_j(x_j) - t_j)^t,
\]

where \(i(\cdot)\) is a mapping \(\{1, \ldots, K\} \to \{1, \ldots, d\}\). The theoretical results of this paper can be generalized in a straightforward way to these types of higher-order basis functions. If the MARS algorithm is applied to the function class generated by \(\phi, \psi\), we refer to this as higher order MARS (HO-MARS).

### 2.2. Faber–Schauder series expansion

The Faber–Schauder system defines a family of functions \([0, 1]^d \to \mathbb{R}\) that can be viewed as the system of indefinite integrals of the Haar wavelet functions. Faber–Schauder functions are piecewise linear in each component and expansion with respect to the Faber–Schauder system results therefore in (componentwise) piecewise linear approximations.

To describe the expansion of a function with respect to the Faber–Schauder system, consider first the univariate case \(d = 1\). Let \(\phi(x) = 1\) and \(\psi = 1(c \in [0, 1/2]) - 1(c \in [1/2, 1])\). The Haar wavelet consists of the functions \(\{\phi, \psi, j = 0, 1, \ldots, k = 0, \ldots, 2^j - 1\}\) with \(\psi_{j,k} = 2^{j/2}\psi(2^j \cdot -k)\). A detailed introduction to Haar wavelets is given for instance in Wojtaszczyk (1997).

Define \(\Phi(x) = \int_0^x \phi(u) \, du = x\) and \(\Psi_{j,k}(x) = \int_0^x \psi_{j,k}(u) \, du\). The functions \(\Psi_{j,k}\) are triangular functions on the interval \([k2^{-j}, (k + 1)2^{-j}]\). The Faber–Schauder system on \([0, 1]\) is \(\{\phi, \Psi, \Phi, j = 0, 1, \ldots; k = 0, \ldots, 2^j - 1\}\).

**Lemma 2.2.** Let \(f\) be a function in the Sobolev space \([f : f' \in L^2[0, 1]]\). Then,

\[
f(x) = f(0) + (f(1) - f(0))x + \sum_{j=0}^{\infty} \sum_{k=0}^{2^j - 1} d_{j,k} \Psi_{j,k}(x)
\]

with \(d_{j,k} = -2^{j/2}(f(k2^{j+1}) - 2f(k2^j/2 + f(k2^{j+1})))\) and convergence in \(L^2[0, 1]\).

**Proof.** It is well-known that the Haar wavelet generates an orthonormal basis of \(L^2[0, 1]\). Since by assumption \(f' \in L^2[0, 1]\),

\[
f'(x) = f(1) - f(0) + \sum_{j=0}^{\infty} \sum_{k=0}^{2^j - 1} d_{j,k} \Psi_{j,k}(x)
\]

with \(d_{j,k} = \int \int f'(u) \psi_{j,k}(u) \, du = -2^{j/2}(f(k2^{j+1}) - 2f(k2^j/2 + f(k2^{j+1})))\) and convergence in \(L^2[0, 1]\). The result follows by integrating both sides together with the fact that \(\|h\|_{L^2[0, 1]} \leq \|h'\|_{L^2[0, 1]}\), whenever \(h' \in L^2[0, 1]\) and \(h(0) = 0\). □

In the univariate case, expansion with respect to the Faber–Schauder system is the same as piecewise linear interpolation on a dyadic grid. On the lowest resolution level, the function \(f\) is approximated by \(f(0) + (f(1) - f(0))x\) which interpolates the
function at $x = 0$ and $x = 1$. If we go up to the $j$th resolution level, the Faber–Schauder reconstruction linearly interpolates the function $f$ on the dyadic grid $x = k/2^{j+1}$, $k = 0, \ldots, 2^{j+1}$. The Faber–Schauder system is consequently also a basis for the continuous functions equipped with the uniform norm. In neural networks terminology, this means that an universal approximation theorem holds.

The multivariate Faber–Schauder system can be obtained by tensorization. Let $Ψ_{-2} := Φ$, $Ψ_{-1} := Φ$, $A_{-1} := \{-2, -1\}$, $(j, k), j = 0, 1, \ldots, c$, $k = 0, 1, \ldots, 2^j - 1$, and $Λ = A_{-∞}$. Then, we can rewrite the univariate Faber–Schauder system on $[0, 1]$ as $Ψ_{i λ} : λ ∈ Λ$. The d-variate Faber–Schauder system on $[0, 1]^d$ consists of the functions 

$$\{Ψ_{i λ} ⊗ \cdots ⊗ Ψ_{i λ_d} : λ_i ∈ Λ\},$$

(cf. Fig. 1). In the following, we write $λ$ for the vector $(λ_1, \ldots, λ_d)$. Given $λ = (λ_1, \ldots, λ_d)$, let $I(λ)$ denote the set containing all indices for which $λ_i \neq -2$. Given an index set $I \subseteq \{1, \ldots, d\}$, $f(I, u_I)$ denotes the function $f$ with components in $f^I = \{1 \ldots d\} \setminus I$ being zero. Moreover, for $I = (i_1, \ldots, i_d)$, $∂_I = ∂_{i_1} \cdots ∂_{i_d}$ stands for the partial derivative with respect to the indices $i_1, \ldots, i_d$. Let also $Ψ_{-1} := Φ$.

**Lemma 2.3.** Suppose that for all sets $I \subseteq \{1, \ldots, d\}$, $∂_I f(0, u_I) \in L^2[0, 1]^d$. Then, with $x = (x_1, \ldots, x_d) \in [0, 1]^d$,

$$f(x) = \sum_{k ∈ A \times \cdots \times A} d_k Ψ_{i λ} (x_1) \cdot \cdots \cdot Ψ_{i λ} (x_d)$$

with $d_k = \int_0^{x_1} \cdots \int_0^{x_d} ∂_I f(0, u_I) du_{i_1} \cdots du_{i_d}$.

The right hand side should be interpreted as $\lim_{t → ∞} \sum_{k ∈ A \times \cdots \times A} Ψ_{i λ} (x_1) \cdot \cdots \cdot Ψ_{i λ} (x_d)$ and converges in $L^2[0, 1]^d$.

**Proof.** By induction on $d$,

$$f(x) = \sum_{i = \{i_1, \ldots, i_d\} \subseteq \{1, \ldots, d\}} \int_0^{x_1} \cdots \int_0^{x_d} ∂_I f(0, u_I) du_{i_1} \cdots du_{i_d}.$$

Now, we can expand each of the partial derivatives with respect to the Haar wavelet

$$∂_I f(0, u_I) = \sum_{k = (λ_1, \ldots, λ_d) ∈ (A \setminus \{-2\})^d} d_k Ψ_{i λ} (u_{i_1}) \cdot \cdots \cdot Ψ_{i λ} (u_{i_d})$$

using that $Ψ_{-1} := Φ$ and $(A \setminus \{-2\})^d = (A \setminus \{-2\}) \times \cdots \times (A \setminus \{-2\})$. Convergence is in $L^2[0, 1]^d$ for the limit $lim_{t → ∞} \sum_{k ∈ (A \setminus \{-2\})^d} Ψ_{i λ} (x_1) \cdot \cdots \cdot Ψ_{i λ} (x_d)$.

Therefore, for $x = (x_1, \ldots, x_d) \in [0, 1]^d$,

$$f(x) = \sum_{I = \{i_1, \ldots, i_d\} \subseteq \{1, \ldots, d\}} \sum_{k = (λ_1, \ldots, λ_d) ∈ (A \setminus \{-2\})^d} d_k Ψ_{i λ} (x_1) \cdot \cdots \cdot Ψ_{i λ} (x_d)$$

using that $Ψ_{-2} = 1$ on $[0, 1]$. □

Because the Haar wavelet functions are piecewise constant, the coefficients $d_k$ can also be expressed as a linear combination of function values $f$. In particular, we recover for $d = 1$ that

$$d_k = 2^{-2/3}(f(x_2) + 2^{2/3} f(x_{2/3}) + f(x_3))$$

Similarly as in the univariate case, the multivariate Faber–Schauder system interpolates the function on a dyadic grid. Given that $f(x) = \sum_{k ∈ A \times \cdots \times A} d_k Ψ_{i λ} (x_1) \cdot \cdots \cdot Ψ_{i λ} (x_d)$, let

$$T_2 f(x) = \sum_{k ∈ A \times \cdots \times A} d_k Ψ_{i λ} (x_1) \cdot \cdots \cdot Ψ_{i λ} (x_d),$$

It can be shown that $T_2 f(x) = f(x)$ for all grid points $x = (x_1, \ldots, x_d)$ with each component $x_i ∈ \{k2^{-1} : k = 0, \ldots, 2^d - 1\}$. Since $T_2 f(x)$ is also piecewise linear in each component, $T_2 f$ coincides with the linear interpolating spline for dyadic knot points. Another consequence is that the universal approximation theorem also holds for the multivariate Faber–Schauder class on $[0, 1]^d$.

**Definition 2.4.** [Faber–Schauder class] For any non-negative integer $M$ and any constant $C > 0$ the $d$-variate Faber–Schauder class truncated at resolution level $M$ is defined as

$$F_{λM}(M, C) = \{x → \sum_{k ∈ A_M × \cdots × A_M} β_k Ψ_{i λ}(x_1) \cdot \cdots \cdot Ψ_{i λ}(x_d) : |β| ≤ C\}.$$

We further assume that $\|f\|_∞ ≤ F$ for all $f ∈ F_{λM}(M, C)$ for some $F > 0$.

The cardinality of the index set $A_M$ is $(1 + 2^M)^d$. The class $F_{λM}(M, C)$ therefore consists of functions with

$$I := (1 + 2^M)^d$$

many parameters. Every function $Ψ_{i λ_1} \cdot \cdots \cdot Ψ_{i λ_d}$ can be represented as a linear combination of $3^d$ MARS functions. This yields the following result.

**Lemma 2.5.** Let $I$ be defined as in (3). Then, for any $M, C > 0$, $F_{λM}(M, C) ⊆ F_{λM}(M, C)$.

**Proof.** For $j = 0, 1, \ldots, k = 0, \ldots, 2^j - 1$,

$$Ψ_{i λ}(x) = 2^{-j/2}(x - k 2^j) - 2^{-j/2+1}(x - k + 1/2 2^j) + 2^{-j/2+1}(x - k + 1 2^j),$$

This shows that any function $f ∈ Ψ_{i λ}(x_1) \cdot \cdots \cdot Ψ_{i λ}(x_d)$ with $λ ∈ A_M × \cdots × A_M$ can be written as a linear combination of at most $3^d$ MARS functions with coefficients bounded by $\|f\|_∞ 2^M/2^d$. □

A consequence is that also the MARS functions satisfy an universal approximation theorem.

**3. Deep neural networks (DNNs)**

Throughout the article, we consider deep ReLU networks. This means that the activation function is chosen as $σ(x) = (x)_+ = \max(x, 0)$, $x ∈ \mathbb{R}$. In computer science, feedforward DNNs are typically introduced via their representation as directed graphs.
For our purposes, it is more convenient to work with the following equivalent algebraic definition. The number of hidden layers or depth of the multilayer neural network is denoted by \( L \). The width of the network is described by a width vector \( \mathbf{p} = (p_0, \ldots, p_{L+1}) \in \mathbb{N}^{L+1} \) with \( p_0 = d \) the input dimension. For vectors \( \mathbf{y} = (y_1, \ldots, y_r) \) and \( \mathbf{v} = (v_1, \ldots, v_r) \in \mathbb{R}^r \), define the shifted activation function \( \sigma_{\varepsilon} : \mathbb{R}^r \rightarrow \mathbb{R}^r \), \( \sigma_{\varepsilon}(\mathbf{y}) = (\sigma(y_1 - \varepsilon), \ldots, \sigma(y_r - \varepsilon)) \).

A network is then any function of the form
\[
f(\mathbf{x}) = W_{L+1} \circ \sigma_{\varepsilon} \circ \cdots \circ W_2 \circ \sigma_{\varepsilon} \circ W_1 \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^d,
\]
with \( W_L \in \mathbb{R}^{p_{L+1} \times p_L} \) and \( v_t \in \mathbb{R}^{p_t} \). For convenience, we omit the composition symbol " \( \circ \) " and write \( f(\mathbf{x}) = W_{L+1} \sigma_{\varepsilon} \cdots W_2 \sigma_{\varepsilon} W_1 \mathbf{x} \). To obtain networks that are real-valued functions, we must have \( p_{L+1} = 1 \).

During the network learning, regularization is induced by combining different methods such as weight decay, batch normalization and dropout. A theoretical description of these techniques seems, however, out of reach at the moment. To derive theory, sparsely connected networks are commonly assumed which are believed to have some similarities with regularized network reconstructions. Under the sparsity paradigm it is assumed that most of the network parameters are zero or non-active. The number of active/non-zero parameters is defined as
\[
s = \sum_{l=1}^{L+1} |W_l|_0 + \sum_{l=1}^{L} |v_l|_0.
\]
where \( |W_l|_0 \) and \( |v_l|_0 \) denote the number of non-zero entries of the weight matrices \( W_l \) and shift vectors \( v_l \). The function class of all networks with the same network architecture \((L, \mathbf{p})\) and network sparsity \( s \) can then be defined as follows.

**Definition 3.1 (ReLU Network Class).** For \( L, s \in \mathbb{N} \), \( F > 0 \), and \( \mathbf{p} = (p_0, \ldots, p_{L+1}) \in \mathbb{N}^{L+2} \) denote by \( \mathcal{F}(L, \mathbf{p}, s) \) the class of \( s \)-sparse ReLU networks \( f \) of the form (4) for which the absolute value of all parameters, that is, the entries of \( W_l \) and \( v_l \), are bounded by one and \( \|f\|_{\infty} \leq F \).

Network initialization is typically done by sampling small parameter weights. If one would initialize the weights with large values, learning is known to perform badly (Goodfellow, Bengio, & Courville, 2016, Section 8.4). In practice, bounded parameters are therefore fed into the learning procedure as a prior assumption via network initialization. We have incorporated this in the function class by assuming that all network parameters are bounded by a universal constant which for convenience is taken to be one here.

Fitting a DNN to data is typically done via stochastic gradient descent of the empirical loss. The practical success of DNNs is partially due to a combination of several methods and tricks that are built into the algorithm. An excellent treatment is (Goodfellow et al., 2016).

4. Deep networks vs. spline type methods

4.1. Improper learning of spline type methods by deep networks

We derive below ReLU network architectures which allow for an approximation of every MARS and Faber–Schauder function on the unit cube \([0, 1]^d\) up to a small error.

**Lemma 4.1.** For fixed input dimension, any positive integer \( M \), any constant \( C \), and any \( \varepsilon \in (0, 1) \), there exists \((L, \mathbf{p}, s)\) with
\[
L = O\left( \log \frac{MC}{\varepsilon} \right), \quad \max_{l=0, \ldots, L} p_l = O(M), \quad s = O\left( M \log \frac{MC}{\varepsilon} \right),
\]
such that
\[
\inf_{h \in \mathcal{F}(L, \mathbf{p}, s)} \|f - h\|_{\infty} \leq \varepsilon \quad \text{for all } f \in \mathcal{F}_{\text{MARS}}(M, C).
\]

Recall that \( f \in \mathcal{F}_{\text{MARS}}(M, C) \) consists of \( M \) basis functions. The neural network \( h \) should consequently also have at least \( O(M) \) active parameters. The previous result shows that for neural networks we need \( O(M \log(M/\varepsilon)) \) many parameters. Using DNNs, the number of parameters is thus inflated by a factor \( \log(M/\varepsilon) \) that depends on the error level \( \varepsilon \).

Recall that \( I = (1 + 2^M)^d \). Using the embedding of Faber–Schauder functions into the MARS function class given in Lemma 2.5, we obtain as a direct consequence the following lemma.

**Lemma 4.2.** For fixed input dimension, any positive integer \( M \), any constant \( C \), and any \( \varepsilon \in (0, 1) \), there exists \((L, \mathbf{p}, s)\) with
\[
L = O\left( \log \frac{C}{\varepsilon} \right), \quad \max_{l=0, \ldots, L} p_l = O(I), \quad s = O\left( I \log \frac{C}{\varepsilon} \right),
\]

such that
\[
\inf_{h \in \mathcal{F}(L, \mathbf{p}, s)} \|f - h\|_{\infty} \leq \varepsilon \quad \text{for all } f \in \mathcal{F}_{\text{MARS}}(M, C).
\]

The number of required network parameters \( s \) needed to approximate a Faber–Schauder function with \( I \) parameters is thus also of order \( I \) up to a logarithmic factor.

The proofs of Lemma 4.1 and Lemma 4.2 are constructive. To find a good initialization of a neural network, we can therefore run MARS in a first step and then use the construction in the proof to obtain a DNN that generates the same function up to an error \( \varepsilon \). This DNN is a good initial guess for the true input–output relation and can therefore be used as initialization for any iterative method.

There exists a network architecture which allows to approximate the product of any two numbers in \([0, 1]\) at an exponential rate with respect to the number of network parameters, cf. Schmidt-Hieber (2017), Yarotsky (2017). This construction is the key step in the approximation theory of DNNs with ReLU activation functions and allows us to approximate the MARS and Faber–Schauder function class. More precisely, we consider the following iterative extension to the product of \( r \) numbers in \([0, 1] \). Notice that a fully connected DNN with architecture \((L, \mathbf{p})\) has \( \sum_{l=0}^{L}(p_{l+1}) = 2^r \) network parameters.

**Lemma 4.3 (Lemma 5 in Schmidt-Hieber (2017)).** For any \( N \geq 1 \), there exists a network \( \text{Multi}^r_N \in \mathcal{F}(N + 5, \lceil\log_2 r\rceil, r, 6r, 6r, \ldots, 6r, 1) \) such that \( \text{Multi}^r_N(\mathbf{x}) = \prod_{j=1}^{r} x_j \) for all \( \mathbf{x} = (x_1, \ldots, x_r) \in [0, 1]^r \).

The total number of network parameters \( s \) is bounded by \( 42r^2(1 + (N + 5)[\log_2 r]) \).

4.2. Lower bounds

In this section, we show that MARS and Faber–Schauder class require much more parameters to approximate the functions \( f(x) = x^2 \) and \( f(x_1, x_2) = (x_1 + x_2 - 1)_+ \). This shows that the converse of Lemma 4.1 and Lemma 4.2 is false and also gives some insights when DNNs work better.

The case \( f(x) = x^2 \): This is an example of a univariate function where DNNs need much fewer parameters compared to MARS and Faber–Schauder class.

**Theorem 4.4.** Let \( \varepsilon > 0 \) and \( f(x) = x^2 \).

(i) If \( M \leq 1/(6\sqrt{F}) \), then
\[
\inf_{g \in \mathcal{F}_{\text{MARS}}(M, \infty)} \|g - f\|_{\infty} \geq \varepsilon.
\]
(ii) If $2^{M+5} \leq 1/\sqrt{\varepsilon}$ then
\[
\inf_{g \in \mathcal{F}_{\text{P}(M, \infty)}} \|g - f\|_{\infty} \geq \varepsilon.
\]
(iii) There exists a neural network $h$ with $O(\log(1/\varepsilon))$ many layers and $O(1/\varepsilon)$ many network parameters such that
\[
\|h - f\|_{\infty} \leq \varepsilon.
\]
(iv) \cite{Yarotsky2017, Theorem 6} Given a network architecture $(L, p)$, there exists a positive constant $c(L)$ such that if $s \leq c(L)\varepsilon^{-1/2}$ then
\[
\inf_{g \in \mathcal{F}(L, p, s)} \|g - f\|_{\infty} \geq \varepsilon.
\]

Part (iii) follows directly from Lemma 4.3. For (iv) one should notice that the number of hidden layers as defined in this work corresponds to $L = 2$ in \cite{Yarotsky2017}. For each of the methods, Theorem 4.4 allows us to find a lower bound on the required number of parameters to achieve approximation error at least $\varepsilon$. For MARS and Faber–Schauder class, we need at least of the order $\varepsilon^{-1/2}$ many parameters whereas there exist DNNs with $O(\log(1/\varepsilon))$ layers that require only $O(\log(1/\varepsilon))$ parameters.

Parts (i) and (ii) rely on the following classical result for the approximation by piecewise linear functions, which for sake of completeness is stated with a proof. Similar results have been used for DNNs by \cite{Yarotsky2017} and \cite{Liang and Srikant2016}, Theorem 11.

**Lemma 4.5.** Denote by $\mathcal{H}(K)$ the space of piecewise linear functions on $[0, 1]$ with at most $K$ pieces and put $f(x) = x^2$. Then,
\[
\inf_{h \in \mathcal{H}(K)} \|h - f\|_{\infty} \geq \frac{1}{8K^2}.
\]

**Proof.** We prove this by contradiction assuming that there exists a function $h^* \in \mathcal{H}(K)$ with $\|h^* - f\|_{\infty} \leq 1/(8K^2)$. Because $h^*$ consists of at most $K$ pieces, there exist two values $0 \leq a < b \leq 1$ with $b - a \geq 1/K$ such that $h^*$ is linear on the interval $[a, b]$. Using triangle inequality,
\[
\begin{align*}
\|h^*(a + b) - (a + b)^2/2\| &\geq \frac{1}{2}h^*(a) + h^*(b) - \frac{(a + b)^2}{2} \geq \frac{a - b}{2} - \frac{1}{2}h^*(a) - a^2 - \frac{1}{2}h^*(b) - b^2 \\
&\geq \frac{1}{4K^2} - \frac{1}{8K^2} = \frac{1}{8K^2}.
\end{align*}
\]
This is a contradiction and the assertion follows. \hfill $\Box$

**Proof of Theorem 4.4(i) and (ii).** A univariate MARS function $f \in \mathcal{F}_{\text{P}(M, \infty)}$ is piecewise linear with at most $M + 1$ pieces. From Lemma 4.5 it follows that $\sup_{x \in [0, 1]} |f(x) - x^2| \geq 1/(8(M + 1)^2) \geq 1/(36M^2)$. This yields (i). To prove (ii), observe that a univariate Faber–Schauder function $g \in \mathcal{F}_{\text{P}(M, \infty)}$ is piecewise linear with at most $2M + 1$ pieces. (ii) thus follows again from Lemma 4.5.

The case $f(x_1, x_2) = (x_1 + x_2 - 1)$: This function can be realized by a neural network with one layer and four parameters. Multivariate MARS and Faber–Schauder functions are, however, built by tensorization of univariate functions. They are not well-suited if the target function includes linear combinations over different inputs as indicated below. Depending on the application this can indeed be a big disadvantage. To describe an instance where this happens, consider a dataset with input being all pixels of an image. The machine learning perspective is that a canonical regression function depends on characteristic features of the underlying object such as edges in the image. Edge filters are discrete directional derivatives and therefore linear combinations of the input. This can easily be realized by a neural network but requires far more parameters if approximated by MARS or the Faber–Schauder class.

**Lemma 4.6.** For $f(x_1, x_2) = (x_1 + x_2 - 1)_+$ on $[0, 1]^2$,
\[
\begin{align*}
(i) \quad &\inf_{h \in \mathcal{F}_{\text{P}(M, \infty)}} \|f - h\|_{\infty} \geq \frac{1}{8(M + 1)}; \\
(ii) \quad &\inf_{h \in \mathcal{F}_{\text{P}(M, \infty)}} \|f - h\|_{L^1} \geq \frac{1}{8M}.
\end{align*}
\]

**Proof.** (i) Let $h \in \mathcal{F}_{\text{P}(M, \infty)}$. The univariate, piecewise linear function $h(x) \in [0, 1] \mapsto \mathbb{R}$ has at most $M$ kinks, where the locations of the kinks do not depend on $x_2$ (some kinks might, however, vanish for some $x_2$). Thus, there exist numbers $0 \leq a < b \leq 1$ with $b - a \geq 1/(M + 1)$, such that $h(x)$ is linear in $[a, b]$ for all $x_2 \in [0, 1]$. Fix $x_2 = 1 - (a + b)/2$, such that $f(x_1, x_2) = a - (a + b)/2$. In particular, $f(x_1, x_2)$ is piecewise linear in $[a, b]$ with one kink, and $f(a, x_2) = f((a + b)/2, x_2) = 0$. Therefore,
\[
\sup_{x_1 \in [a, b]} |f(x_1, x_2) - h(x_1, x_2)| \geq \frac{b - a}{8} \geq \frac{1}{8(M + 1)}.
\]
(ii) Similar to (i) with $a = 0$ and $b = 2^{-m-1}$. \hfill $\Box$

**5. Risk comparison for nonparametric regression**

In nonparametric regression with random design the aim is to estimate the dependence of a response variable $Y \in \mathbb{R}$ on a random vector $X \in [0, 1]^d$. The dataset consists of $n$ independent copies $(Y_i, X_i)_{i=1,...,n}$ generated from the model
\[
Y_i = f_0(X_i) + e_i, \quad i = 1, \ldots, n.
\]
We assume that $f_0 : [0, 1]^d \rightarrow [0, F]$ for some $F > 0$ and that the noise variables $e_i$ are independent, standard normal and independent of $(X_i)$. An estimator or reconstruction method will typically be denoted by $\hat{f}$. The performance of an estimator is measured by the prediction risk
\[
R(\hat{f}, f_0) \equiv \mathbb{E}[\hat{f}(X) - f_0(X_i)]^2,
\]
with $X$ having the same distribution as $X_1$ but being independent of the sample $(Y_i, X_i)$.

The cross-entropy in this model coincides with the least-squares loss, see \cite{Goodfellow2016}, p. 129. A common property of all considered methods in this article is that they aim to find a function in the respective function class minimizing the least squares loss. Gradient methods for network reconstructions will typically get stuck in a local minimum or a flat saddle point. We define the following quantity that determines the expected difference between the loss induced by a reconstruction method $\hat{f}$ taking values in a class of networks $\mathcal{F}$ and the empirical risk minimizer over $\mathcal{F}$,
\[
\Delta_r(\hat{f}_n, f_0, \mathcal{F}) := \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} (Y_i - f_0(X_i))^2 - \argmin_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (Y_i - f(X_i))^2\right].
\]

The quantity $\Delta_r(\hat{f}_n, f_0, \mathcal{F})$ allows us to separate the statistical risk induced by the selected optimization method. Obviously $\Delta_r(\hat{f}_n, f_0, \mathcal{F}) \geq 0$ and $\Delta_r(\hat{f}_n, f_0, \mathcal{F}) = 0$ if $\hat{f}_n$ is the empirical risk minimizer over $\mathcal{F}$. For convenience, we write $\Delta_r := \Delta_r(\hat{f}_n, f_0, \mathcal{F}(L, p, s))$. 

\[\text{(6)}\]
Theorem 5.1. Consider the d-variate nonparametric regression model (5) with bounded regression function $f_0$. For any positive integer $M$, and any constant $C$, there exists a neural network class $\mathcal{F}(L, p, s)$ with $L = O(\log M n)$, $\max p_i = O(M)$ and $s = O(M \log M n)$, such that for any neural network estimator $\hat{f}$ with values in the network class $\mathcal{F}(L, p, s)$ and any $0 < \gamma \leq 1$,

$$R(\hat{f}, f_0) \leq (1 + \gamma)^2 \left( \inf_{f \in \mathcal{F}_{\mathcal{F}}(L, p, s)} \mathbb{E}[\{(f(X) - f_0(X))^2\}^2] + \Delta_n \right) + O \left( \frac{M \log^2(M n) \log(M + 1)}{n} \right).$$

(7)

Theorem 5.1 gives an upper bound for the risk of the neural network estimator in terms of the best approximation over all MARS functions plus a remainder. The remainder is roughly of order $\Delta_n^2 + M / n$, where $M$ is the maximal number of basis functions in $\mathcal{F}_{\mathcal{F}}(M, C)$. If $M$ is small the approximation term dominates. For large $M$, the remainder can be of larger order. This does, however, not imply that neural networks are outperformed by MARS. It is conceivable that the risk of the MARS estimator also includes a variance term that scales with the number of parameters over the sample size and is therefore of order $M / n$. To determine an $M$ that balances both terms is closely related to the classical bias–variance trade-off and depends on properties of the true regression function $f$. For a result of the same flavor, see Theorem 1 in Zhang et al. (2017). A similar result also holds for the Faber–Schauder approximation.

Theorem 5.2. Consider the d-variate nonparametric regression model (5) with bounded regression function $f_0$. For any positive integer $M$, and any constant $C$, there exists a neural network class $\mathcal{F}(L, p, s)$ with $L = O(\log M n)$, $\max p_i = O(1)$ and $s = O(1 \log M n)$, such that for any neural network estimator $\hat{f}$ with values in the network class $\mathcal{F}(L, p, s)$ and any $0 < \gamma \leq 1$,

$$R(\hat{f}, f_0) \leq (1 + \gamma)^2 \left( \inf_{f \in \mathcal{F}_{\mathcal{F}}(L, p, s)} \mathbb{E}[\{(f(X) - f_0(X))^2\}^2] + \Delta_n \right) + O \left( \frac{I \log^2(n) \log(I + 1)}{n^\gamma} \right).$$

6. Finite sample results

This section provides a simulation-based comparison of DNNs, MARS and expansion with respect to the Faber–Schauder system. We also consider higher-order MARS (HO-MARS) using the function system (2). In particular, we study frameworks in which the higher flexibility of networks yields much better reconstructions compared to the other approaches. For MARS we rely on the py-earth package in Python. DNNs are fitted using the Adam optimizer in Keras (Tensorflow backend) with learning rate 0.001 and decay 0.00021.

As it is well-known the network performance relies heavily on the initialization of the network parameters. In the settings below, several standard initialization schemes resulted in suboptimal performance of deep networks compared with the spline based methods. One reason is that the initialization can ‘deactivate’ many units. This means that we generate $w, v$ such that for every incoming signal $x$ occurring for a given dataset, $(w \cdot x + v_i) = 0$. The gradients for $w, v$ are then zero and the unit remains deactivated for all times reducing the expressiveness of the learned network. For one-dimensional inputs with values in $[0, 1]$, for instance, the $j$th unit in the first layer is of the form $(w_j x + v_j)$. The unit has then the kink point in $x = -v_j / w_j$. The unit is deactivated, in the sense above, if $-v_j / w_j > 1$ and $w_j > 0$ or if $-v_j / w_j < 0$ and $w_j < 0$. To increase the expressiveness of the network the most desirable case is that the kink falls into the interval $[0, 1]$. In view of the inequalities above this requires that $v_j$ and $w_j$ do not have the same sign and $|w_j| \geq |v_j|$. A similar argument holds also for the deeper layers.

The Glorot/Xavier uniform initializer suggested in Glorot and Bengio (2010) initializes the network by setting all shifts/biases to zero and sampling all weights on layer $\ell$ independently from a uniform distribution $U[-b_\ell, b_\ell]$ with $b_\ell = \sqrt{6/(M_{\ell-1} + M_\ell)}$ and $M_\ell$ the number of nodes in the $\ell$-th layer. This causes the kink to be at zero for all units. To distribute the kink location more uniformly, we also sample the entries of the $\ell$-th shift vector independently from a uniform distribution $U[-b_\ell, b_\ell]$.

It is also possible to incorporate shape information into the network initialization. If the weights are generated from a uniform distribution $U[0, b_\ell]$, all slopes are non-negative. This can be viewed as incorporating prior information that the true function increases at least linearly. In view of the argument above, it is then natural to sample the biases from $U[-b_\ell, 0]$. We call this the increasing Glorot initializer.

As described above, it might still happen that the initialization scheme deactivates many units, which might lead to extremely bad reconstructions. For a given dataset, we can identify these cases because of their large empirical loss. As often done in practice, for each dataset we re-initialize the neural network several times and pick the reconstruction with the smallest empirical loss.

The case $f_0(x) = x^2$: In the previous section, we have seen that MARS and Faber–Schauder have worse approximation properties for the function $f_0(x) = x^2$ if compared with DNNs. It is therefore of interest to study this as a test case with simulated independent observations from the model

$$Y_i = \epsilon_i, \quad X_i \sim U[0, 1], \quad \epsilon_i \sim N(0, 0.2), \quad i = 1, \ldots, 1000. \quad (8)$$

Since $f_0(x) = x^2$ is smooth, a good approximation should already be provided for a relatively small number of parameters in any of the considered function classes. Thus, we set $M = 10$ for MARS and $K = 3, M = 10$ for higher order MARS allowing for polynomials of degree up to three. For the Faber–Schauder class $FS_M(L, C)$ we set $M = 3$ meaning that details up to size $2^{-3}$ can be resolved. For the neural network reconstruction, we consider dense networks with three hidden layers and five units in each hidden layer. The network parameters are initialized using the increasing Glorot initializer described above. The simulation results are summarized in Table 1, see also Fig. 2. Concerning the prediction risk, all methods perform equally well, meaning that the effect of the additive noise in model (8) dominates the approximation properties of the function classes. We find that higher order MARS often yields a piecewise linear approximation of the regression function and does not include any quadratic term.

It is natural to ask about the effect of the noise on the reconstruction in model (8). We therefore also study the same problem without noise, that is,

$$Y_i = \epsilon_i, \quad X_i \sim U[0, 1], \quad i = 1, \ldots, 1000. \quad (9)$$

Results are summarized in Table 1. As the underlying function class contains the square function, higher order MARS finds in this case the exact representation of the regression function. MARS, Faber–Schauder and neural networks result in a much smaller prediction risk if compared to the case with noise. DNNs and Faber–Schauder series estimation are of comparable quality while MARS performs worse.

The theory suggests that for squared regression function, DNNs should also perform better than Faber–Schauder. We believe that DNNs do not find a good local minimum in most cases explaining the discrepancy between practice and applications. For further improvements of DNNs, a more refined network initialization scheme is required.

The case $f_0(x) = \sin(5 \cdot 2\pi x)$: To compare the different methods for rough functions, we simulate independent observations from

$$Y_i = \sin(5 \cdot 2\pi X_i), \quad X_i \sim U[0, 1], \quad i = 1, \ldots, 1000. \quad (10)$$
show that in this problem neural networks and DNNs need much fewer parameters than MARS or the Faber–Schauder class for this function. We generate independent observations from

\[ Y_i = \begin{cases} -\frac{1}{2}, & \text{if } X_i < \frac{1}{2}, \\ \frac{1}{2}, & \text{if } X_i \geq \frac{1}{2}, \end{cases} \quad X_i \sim \mathcal{U}[0, 1], \quad i = 1, \ldots, 1000. \tag{11} \]

We use the same choices of parameters as in the simulation before. Results are shown in Table 2 and Fig. 3, lower parts. Compared with the other methods, the neural network reconstruction does not add spurious artifacts around the jump location.

**The case** \( f_0(x) = -\frac{1}{2} + 1(x \geq \frac{1}{2}); \) As another toy example we consider regression with a jump function to investigate the approximation performance in a setup in which standard smoothness assumptions are violated. For this we generate independent observations from

\[ Y_i = \begin{cases} \frac{-1}{2}, & \text{if } X_i < \frac{1}{2}, \\ \frac{1}{2}, & \text{if } X_i \geq \frac{1}{2}, \end{cases} \quad X_i \sim \mathcal{U}[0, 1], \quad i = 1, \ldots, 1000. \tag{11} \]

Notice that the regression function \( f_0 \) is piecewise linear with change of slope along the line \( x_1 + x_2 = 1 \), \( \mathbf{x} = (x_1, x_2)^T \in [0, 1]^2 \). For convenience we only consider the more flexible higher order MARS. The prediction risks reported in Table 3 and the reconstructions displayed in Fig. 4 show that in this problem neural networks outperform higher order MARS and reconstruction with respect to the Faber–Schauder class even if we allow for more parameters.

**Dependence on input dimension:** In this simulation, dependence of the risk on the input dimension is studied. We generate independent observations from

\[ Y_i = \left( \sum_{j=1}^{d} X_{i,j}^2 - \frac{d}{3} \right)_{+}, \quad \mathbf{x} = (X_{i,1}, \ldots, X_{i,d})^T \sim \mathcal{U}[0, 1]^d, \quad i = 1, \ldots, 1000. \tag{13} \]

This function is constructed in such a way that it can be well-approximated by both higher order MARS and neural networks. Independently of the dimension \( d \) the setup for higher order MARS is \( K = 10 \) and \( M = 30 \). We choose a densely connected neural network with architecture \( L = 15, p = (d, 10, \ldots, 10, 1) \) and Glorot uniform initializer. The prediction risks for \( d = 1, 2, 5, 10 \) are summarized in Table 4. The results show that both methods suffer from the curse of dimensionality. Neural networks achieve a consistently smaller risk for all dimensions.
To study the approximation performance for large input dimensions we sample independent observations from the model

\[ Y_i = \left( \sum_{j=1}^{10} X_{ij}^2 - \frac{10}{3} \right)_+, \quad X_i = (X_{i1}, \ldots, X_{id})^\top \sim U[0, 1]^d, \]

\[ i = 1, \ldots, 1000. \]  

(14)

Here, the number of active parameters 10 is fixed and small in comparison to the input dimension. For \( d = 1000 \) the input dimension is the same as the sample size. We consider higher order MARS and neural networks with the same parameters as before. The risks are summarized in Table 5. The obtained values are comparable with the results in the last row of Table 4 showing that both neural network and higher order MARS reconstructions are adaptive with respect to the active parameters of the model.

**Acknowledgments**

We are very grateful to two referees and an associate editor for their constructive comments, which led to substantial improvement of an earlier version of this manuscript.

**Appendix. Proofs**

**Proof of Lemma 4.1.** We first consider the approximation of one basis function \( h_m \) for fixed \( m \in \{1, \ldots, M\} \) by a network. By (1), each function \( h_m \) has the representation

\[ h_m(x) = \prod_{j \in I} (s_j x_j - t_j)_+ \]

for some index set \( I \subseteq \{1, \ldots, d\} \), sign vectors \( s_j \in \{-1, 1\} \) and shifts \( t = (t_j)_{j \in I} \in [0, 1]^{|I|} \).

Now we use Lemma 4.3 to construct a network

\[ H_{m,N} \in \mathcal{F}\left( (N + 5) \lceil \log d \rceil + 1, (d, d, 6d, \ldots, 6d, 1), s \right) \]
such that \( |H_{m,N} - h|_\infty \leq 3^d 2^{-N} \). The first hidden layer of the network \( H_{m,N} \) builds the factors \((s_j(x_i - t_j))_j\) for the input \( x \in \mathbb{R}^d \). The remaining \( d - 1 \) layers of the first term are set to one. This can be achieved by choosing the first weight matrix to be \( W_1 = (s_j(i = j, i \in \{1, \ldots, d\}) \) and the corresponding shift vector as \( v_1 = (s_j(1 \in \{1, \ldots, d\}) \). The first hidden layer of the network \( H_{m,N} \) builds the factors \((s_j(x_i - t_j))_j\) for the input \( x \in \mathbb{R}^d \). The remaining \( d - 1 \) layers of the first term are set to one. This can be achieved by choosing the first weight matrix to be \( W_1 = (s_j(i = j, i \in \{1, \ldots, d\}) \) and the corresponding shift vector as \( v_1 = (s_j(1 \in \{1, \ldots, d\}) \).

Lemma A.2. Let \( m \) be an integer and \( (s_j(x_i - t_j))_j \) be \( \delta \)-covering numbers of \( \mathcal{F}(N + 5)[\log_2 d] + 1 \). Then, there exists a network \( H_{m,N} \) such that \( H_{m,N} = \mathcal{F}(N + 5)[\log_2 d] + 2 \).

Theorem 5.1. Consider a d-variate nonparametric regression model (5) with unknown regression function \( f \). Let \( \hat{f} \) be any estimator taking values in \( \mathcal{F} \) and \( f \) be any function in \( \mathcal{F} \). If \( \mathcal{F} \) is as defined in (6) and \( N \) is the number of \( \mathcal{F} \) balls with radius \( \delta \) that covers \( \mathcal{F} \), then, there exists a universal constant \( K \) such that for any \( \delta \), \( \gamma \), \( \epsilon \), \( \delta_0 \) as in (6),

\[
\mathbb{E} \left[ f(X) - \hat{f}(X) \right] \leq \mathbb{E} \left[ f(X) - \hat{f}(X) \right] + \frac{M \log(\text{log}(\text{log}(\text{log}(\text{log}(\text{log}(d))))))}{n}.
\]

Theorem 5.2. For replacing \( M \) in the bounds, the proof can be derived in the same way as the proof of Theorem 5.1.

**References**


Columbia University, New York, New York, USA: PMLR.


