

University of Münster  
Faculty of Mathematics and Computer Science

# Explicit Construction of Deep Neural Networks

Fabian Bremer  
Matriculation Number: 357260

**Master Thesis**  
for obtaining the degree of Master of Science

Primary Reviewer: Prof. Dr. Mario Ohlberger  
Secondary Reviewer: Prof. Dr. Christian Engwer

Date of Submission: June 5, 2023



# Contents

<b>1</b>	<b>Motivation and Preliminaries</b>	<b>2</b>
1.1	Motivation . . . . .	2
1.2	Introduction . . . . .	2
1.3	Preliminaries . . . . .	3
1.3.1	Deep Neural Networks . . . . .	3
1.3.2	Further Definitons and Notation . . . . .	6
<b>2</b>	<b>Main Results</b>	<b>8</b>
2.1	Approximation by Chebyshev Polynomials . . . . .	8
2.1.1	Expansion . . . . .	8
2.1.2	Interpolation . . . . .	10
2.1.3	Interpolation Based on Approximate Function Values . . . . .	11
2.2	ReLU DNN Approximation of Chebyshev Polynomials . . . . .	16
2.2.1	Multiplication by DNNs . . . . .	16
2.2.2	Approximation of Chebyshev Polynomials of the First Kind . . . . .	21
2.2.3	Approximation of Tensor Product Chebyshev Polynomials . . . . .	32
2.3	Construction of ReLU DNN Approximations of Multivariate Holo- morphic Functions . . . . .	36
<b>3</b>	<b>Implementation</b>	<b>40</b>
3.1	Implementing the Chebyshev Coefficients . . . . .	40
3.2	Constructing the DNNs . . . . .	42
3.2.1	Constructing the DNNs for Multiplication . . . . .	43
3.2.2	Constructing the DNNs for univariate and multivariate Cheby- shev Polynomials . . . . .	45
3.2.3	Algorithm for the Construction of a DNN for approximating a given Function . . . . .	47
<b>4</b>	<b>Numerical Results</b>	<b>50</b>
4.1	Approximation of Univariate Functions . . . . .	50
4.1.1	Comparison with trained DNNs . . . . .	56
4.2	Approximation of Multivariate Functions . . . . .	62
<b>5</b>	<b>Comparisons</b>	<b>65</b>
<b>6</b>	<b>Conclusion</b>	<b>68</b>
6.1	Summary . . . . .	68
6.2	Discussion and Outlook . . . . .	68
	<b>Literature</b>	<b>70</b>

---

# 1 Motivation and Preliminaries

## 1.1 Motivation

Deep Neural Networks (DNNs) are a powerful tool in the approximation of functions and have gained a lot of public interest as the foundation of so called *Artificial Intelligence (AI)* applications. They have been used in a wide range of implementations, such as image recognition, speech recognition, natural language processing and many more (cf. [14], [15], [9]). However, the theoretical understanding of DNNs is still in its infancy.

While the mathematical structure of DNNs as repeated concatenations of affine mappings and non-linear activation functions is well-studied, there are still many unanswered questions regarding their behavior and capabilities. Understanding the theoretical foundations of DNNs is therefore a crucial building stone in the development of further DNN applications, such as AI systems.

One promising approach to understanding DNNs is the constructive approach. Unlike traditional approaches that seek to unravel the black-box nature of these models, the constructive approach focuses on building a mathematical understanding from the ground up. It aims to explore the theoretical foundations of DNNs by systematically constructing and analyzing simpler neural network architectures, investigating their properties, and leveraging insights gained to unravel the complexity of more complex DNNs.

## 1.2 Introduction

This thesis is concerned with the construction of DNNs that emulate multivariate Chebyshev polynomials and using these DNNs to approximate a large class of functions. Importantly, the proofs of the results will be constructive, i.e. the DNNs will be explicitly constructed and will not depend on training by loss minimization algorithms. These results, developed in Chapter 2 of this thesis, are mainly based on the paper *Constructive Deep ReLU Neural Network Approximation*, [10], by Lukas Herrmann, Joost A.A. Opschoor and Christoph Schwab.

After some definitions, notations and preliminary results, Chapter 2 will introduce the theory leading to the construction of DNNs that emulate univariate Chebyshev polynomials in Lemma 2.7. This will be a foundation for the main results, Theorem 2.8 - the construction of tensor product Chebyshev polynomials as DNNs - and Theorem 2.9 - the application of these DNNs in the approximation of a large class of functions. These results will then be implemented in Chapter 3, where the path from theory to actual coding will be shown. The results of the implementation

will be presented in Chapter 4, where the constructed DNNs will be tested and compared to trained DNNs. Chapter 5 will contextualize the results by highlighting a few other publications in this area and the thesis will be concluded with a brief summary and an outlook in Chapter 6.

The main contributions of this thesis are on the one hand proving the error bound and the bounds on depth and size in Lemma 2.7, since this Lemma has only been postulated, but not proven in [10]<sup>1</sup> and on the other hand the implementation of the results, as the authors of [10] only provide a theoretical proof, but no actual implementation has been made public. The implementation will be done in Python, using the framework PyTorch [22].

### 1.3 Preliminaries

The following chapters will give the definitions and notations used in the rest of the thesis as well as state some preparatory results. Firstly Deep Neural Networks will be introduced and some of their properties discussed. Subsequently some equivalent definitions for univariate Chebyshev polynomials will be given, multivariate Chebyshev polynomials introduced and some notation for the remaining chapters presented.

#### 1.3.1 Deep Neural Networks

**Definition 1.1.** A **Deep Neural Network (DNN)** is a repeated concatenation of affine mappings and a specific non-linear map, called activation function  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  of the DNN. Define  $\sigma(x) = \max\{0, x\}$ ,  $x \in \mathbb{R}$  as the rectified linear unit (ReLU) activation function. The DNN consists of a fixed number of hidden layers  $L \in \mathbb{N}_0$  and numbers  $N_\ell \in \mathbb{N}$  of so called *computation nodes* in layer  $\ell \in \{1, \dots, L+1\}$ .  $N_0$  is the input dimension of the DNN and  $N_{L+1}$  is the output dimension. The map  $\Phi : \mathbb{R}^{N_0} \rightarrow \mathbb{R}^{N_{L+1}}$  is said to be realized by (and will therefore be called) a *feedforward* neural network, if for certain weights  $A_{i,j}^\ell \in \mathbb{R}$ , and biases  $b_j^\ell \in \mathbb{R}$  it holds for all  $x = (x_i)_{i=1}^{N_0}$ , that:

$$w_j^1 := \sigma \left( \sum_{i=1}^{N_0} A_{i,j}^1 x_i + b_j^1 \right), \quad j \in \{1, \dots, N_1\}$$

and

$$w_j^{\ell+1} := \sigma \left( \sum_{i=1}^{N_\ell} A_{i,j}^{\ell+1} w_i^\ell + b_j^{\ell+1} \right), \quad \ell \in \{1, \dots, L-1\}, \quad j \in \{1, \dots, N_{\ell+1}\}$$

---

<sup>1</sup>The proof is promised to be elaborated in the dissertation of one of the coauthors, expected to be published in the near future.

and finally

$$\Phi(x) = (w_j^{L+1})_{j=1}^{N_{L+1}} = \left( \sum_{i=1}^{N_L} A_{i,j}^{L+1} w_i^L + b_j^{L+1} \right)_{j=1}^{N_{L+1}}.$$

The number of hidden layers  $L$  of a DNN is referred to as its depth, denoted by  $\text{depth}(\Phi)$ . If  $L = 0$ , then the previous equation holds with  $w_i^0 := x_i$  for  $i = 1, \dots, N_0$ . Such DNNs of depth 0 realize affine functions<sup>2</sup>. Define the total number of nonzero weights and biases as the size of the DNN, i.e.

$$\text{size}(\Phi) := |\{(i, j, \ell) : A_{i,j}^\ell \neq 0\}| + |\{(j, \ell) : b_j^\ell \neq 0\}|.$$

Let  $\text{size}_{\text{in}}(\Phi)$  and  $\text{size}_{\text{out}}(\Phi)$  be the number of nonzero weights and biases in the input resp. the output layer of  $\Phi$ , i.e.  $\text{size}_{\text{in}}(\Phi) := |\{(i, j) : A_{i,j}^1 \neq 0\}| + |\{j : b_j^1 \neq 0\}|$  and  $\text{size}_{\text{out}}(\Phi) := |\{(i, j) : A_{i,j}^{L+1} \neq 0\}| + |\{j : b_j^{L+1} \neq 0\}|$ , where  $A$  and  $b$  are the weights and biases.

For brevity say that  $A \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$  are the weights of layer  $\ell \in \{1, \dots, L+1\}$  if  $A_{ji} = A_{i,j}^\ell$ . A DNN of depth 0 with weights  $A$  and bias  $b$  will be denoted by  $((A, b))$ .

From Definition 1.1 the following definitions and associated claims are easily deduced.

**Definition 1.2. Parallelization and Concatenation of DNNs:**

- Let  $L \in \mathbb{N}_0$  be the depth of two DNNs  $\Phi_1$  and  $\Phi_2$ , that both also have the same input dimension  $d$ . Let  $n_i$  be the output dimension of  $\Phi_i$  for  $i \in \{1, 2\}$ . The map

$$(\Phi_1, \Phi_2) : \mathbb{R}^d \rightarrow \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} : x \mapsto (\Phi_1(x), \Phi_2(x))$$

is called **parallelization** of  $\Phi_1$  and  $\Phi_2$ . It is also a DNN (that can be understood as emulating the DNNs  $\Phi_1$  and  $\Phi_2$  parallelly) and fulfills the following properties:

$$\text{depth}((\Phi_1, \Phi_2)) = L, \quad \text{size}((\Phi_1, \Phi_2)) = \text{size}(\Phi_1) + \text{size}(\Phi_2).$$

This can easily be verified from the definition or confirmed in [23] or - in a more general statement for more than two DNNs - in [12].

- The concept of parallelization can be broadened to DNNs with different input dimensions: Let  $\Phi_1$  be a DNN with input dimension  $n_1$  and output dimension

---

<sup>2</sup>Note that in literature (see for example [4], [5] or [18]) these are often referred to as *shallow* in contrast to the *deep* NNs with depth  $> 0$ . However, since this distinction is not suitable for the purpose of this thesis, the term *DNN* will be used for all kinds of neural networks.

$m_1$  and  $\Phi_2$  be a DNN with input dimension  $n_2$  and output dimension  $m_2$ . Let  $L \in \mathbb{N}_0$  be the depth of both DNNs. The DNN  $(\Phi_1, \Phi_2)_d$  is called **full parallelization of networks with distinct inputs** of  $\Phi_1$  and  $\Phi_2$ :

$$(\Phi_1, \Phi_2)_d : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \rightarrow \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} : (x, x') \mapsto (\Phi_1(x), \Phi_2(x')).$$

It holds that

$$\text{depth}((\Phi_1, \Phi_2)_d) = L, \quad \text{size}((\Phi_1, \Phi_2)_d) = \text{size}(\Phi_1) + \text{size}(\Phi_2).$$

See [6] for a detailed construction of these DNNs.

- Let  $\Phi_1$  be a DNN with input dimension  $d$  and output dimension  $n$  and  $\Phi_2$  be a DNN with input dimension  $m$  and output dimension  $d$ . The **concatenation of  $\Phi_1$  and  $\Phi_2$**  is defined by the map

$$\Phi_1 \circ \Phi_2 : \mathbb{R}^m \rightarrow \mathbb{R}^n : x \mapsto \Phi_1(\Phi_2(x)),$$

which is again a DNN with the properties

$$\text{depth}(\Phi_1 \circ \Phi_2) = \text{depth}(\Phi_1) + \text{depth}(\Phi_2) - 1$$

and

$$\text{size}(\Phi_1 \circ \Phi_2) \leq 2\text{size}(\Phi_1) + 2\text{size}(\Phi_2).$$

This again can be deducted directly from the definition or verified in [23]. Even more rigorous - though for this thesis not necessary - results regarding the concatenation of DNNs can be found in [12].

- For all  $n \in \mathbb{N}$  and  $L \in \mathbb{N}_0$  the **identity network**  $\Phi_{n,L}^{\text{Id}}$  with

$$\Phi_{n,L}^{\text{Id}} : \mathbb{R}^n \rightarrow \mathbb{R}^n : x \mapsto x$$

exists and has the properties

$$\text{depth}(\Phi_{n,L}^{\text{Id}}) = L, \quad \text{size}(\Phi_{n,L}^{\text{Id}}) \leq 2n(L + 1).$$

Construction of these DNNs and verification of these claims is again not difficult (see for example [6]).

### 1.3.2 Further Definitons and Notation

**Definition 1.3.** Let  $n \in \mathbb{N}_0$ . Define the univariate  $n$ -th **Chebyshev polynomial** of the first kind as  $T_n$ , such that

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Equivalently define:

$$T_n(\cos(\theta)) = \cos(n\theta) \Leftrightarrow T_n(x) = \begin{cases} \cos(n \arccos x) & \text{if } |x| \leq 1 \\ \cosh(n \operatorname{arcosh} x) & \text{if } x \geq 1 \\ (-1)^n \cosh(n \operatorname{arcosh}(-x)) & \text{if } x \leq -1 \end{cases}$$

It is easy to verify the equivalence of both definitions (cf. e.g. [29]).

The recursive definition can be generalized to:

$$\forall m, n \in \mathbb{N}_0 : \quad T_{m+n} = 2T_m T_n - T_{|m-n|},$$

since (w.l.o.g.  $m > n$ ) by the rules of addition of cosines it holds that:

$$\begin{aligned} \cos((n+m)\theta) &= \cos\{n\theta + m\theta\} \\ &= \cos(n\theta) \cos(m\theta) - \sin(n\theta) \sin(m\theta), \\ \cos((m-n)\theta) &= \cos\{m\theta - n\theta\} \\ &= \cos(m\theta) \cos(n\theta) + \sin(m\theta) \sin(n\theta), \end{aligned}$$

and therefore

$$\begin{aligned} \cos((n+m)\theta) + \cos((m-n)\theta) &= 2 \cos(m\theta) \cos(n\theta) \\ \Leftrightarrow \cos((n+m)\theta) &= 2 \cos(m\theta) \cos(n\theta) - \cos((m-n)\theta), \end{aligned}$$

from which the claim follows by the above definition.

For  $K \in \mathbb{N}$  and  $k = (k_j)_{j=1}^K \in \mathbb{N}_0^K$ , denote **tensor product Chebyshev polynomials** by

$$T_k(x) := \prod_{j=1}^K T_{k_j}(x_j),$$

for  $x = (x_j)_{j=1}^K \in [-1, 1]^K$ .

**Definition 1.4.** Let  $(X, \Sigma, \mu)$  be a measure space and let  $f : X \rightarrow \mathbb{R}$  be a measurable function. A number  $a \in \mathbb{R} \cup \{+\infty\}$  is called an **essential upper bound** of  $f$  if the measurable set  $f^{-1}(a, \infty)$  is a set of  $\mu$ -measure zero, i.e. if  $f(x) \leq a$  for  $\mu$ -almost all  $x$  in  $X$ . Let

$$U_f^{\text{ess}} = \{a \in \mathbb{R} : \mu(f^{-1}(a, \infty)) = 0\}$$

be the set of essential upper bounds. Then the **essential supremum** is defined as

$$\text{ess sup } f = \begin{cases} \inf U_f^{\text{ess}}, & \text{if } U_f^{\text{ess}} \neq \emptyset, \\ +\infty, & \text{otherwise.} \end{cases}$$

**Notation 1.5.** The following notation will be used throughout this thesis.

1. For  $k \in \mathbb{N}_0$  and a subset  $S \subset \mathbb{N}_0$ , define  $\mathbb{1}_S(k) := 1$  if  $k \in S$ , and  $\mathbb{1}_S(k) := 0$  otherwise.
2. For  $K \in \mathbb{N}$  and  $k \in \mathbb{N}_0^K$ , define  $\mathbb{1}_S(k) := \sum_{j=1}^K \mathbb{1}_S(k_j)$  and denote by  $|k|_0 := \mathbb{1}_{\mathbb{N}}(k)$  the number of nonzero components of  $k$ .
3. For finite index sets  $\Lambda \subset \mathbb{N}_0^K$ , denote the number of elements by  $|\Lambda|$  and the maximum coordinatewise degree by  $m_\infty(\Lambda) := \max_{k \in \Lambda} \|k\|_{\ell^\infty}$ .
4. Error estimates may be expressed in terms of the  $W^{1,\infty}$ -Sobolev norm, which is defined, for an open and bounded domain  $\Omega \subset \mathbb{R}^K$ , as  $\|u\|_{W^{1,\infty}(\Omega)} = \max \left\{ \|u\|_{L^\infty(\Omega)}, \max_{i=1}^K \left\| \frac{\partial}{\partial x_i} u \right\|_{L^\infty(\Omega)} \right\}$ ,  $\frac{\partial}{\partial x_i}$  denoting weak derivatives. For integer  $s \geq 1$ , the  $W^{s,\infty}$ -norm is defined analogously, taking the maximum over all weak derivatives of order at most  $s$ .
5. A superscript tilde (e.g.  $\tilde{f}$ ) denotes a DNN. A superscript breve (e.g.  $\breve{f}$ ) denotes a corrupted quantity. Typically,  $\breve{f}$  denotes a numerical approximation of the map  $f$ , due to some measurement or due to some discretization error in approximating the map  $f$ .
6. The space of polynomials of degree at most  $p$  is denoted by  $\mathbb{P}_p$  for  $p \geq 0$ . The space of polynomials in  $K \in \mathbb{N}$  variables of coordinatewise degree at most  $p$  is denoted by  $\mathbb{Q}_p := \text{span} \{x^\nu : \nu \in \mathbb{N}_0^K \text{ and } \|\nu\|_{\ell^\infty} \leq p\}$ .

**Definition 1.6.** For circles in the complex plane the following notation is used: For all  $r > 0$  define  $\Gamma_r := \{z \in \mathbb{C} : |z| = r\}$ . With this define: For all  $\rho \geq 1$ , the closed **Bernstein ellipse**:

$$\mathcal{E}_\rho := \left\{ \frac{z + z^{-1}}{2} \in \mathbb{C} : 1 \leq |z| \leq \rho \right\}.$$

This can be understood as the image of the annulus  $\{z \in \mathbb{C} : 1 \leq |z| \leq \rho\}$  under the map  $z \mapsto \frac{z+z^{-1}}{2}$ .

For higher dimensions define the isotropic **Bernstein polyellipse**  $\mathcal{E}_\varrho = \mathcal{E}_\rho^K$  with polyradius  $\varrho = (\rho, \dots, \rho) \in \mathbb{R}^K$  for some  $\rho \in (1, \infty)$  and some  $K \in \mathbb{N}$ .



## 2 Main Results

### 2.1 Approximation by Chebyshev Polynomials

Chebyshev polynomials as introduced in Definition 1.3 were first studied by Pafnuty Chebyshev<sup>3</sup> in the 19th century [1]. They have since been of interest in different fields of mathematics, among others in numerics, due to various properties they possess, such as the distribution of their roots, which provide grid points for polynomial interpolation with a small Lebesgue constant<sup>4</sup> that outperform interpolation on equidistant grid points in many cases, and that the set of Chebyshev polynomials is closed under composition [26]. In approximation theory it is noted that, due to the behavior of the error produced by Chebyshev approximation, it is close to the *Best Approximating Polynomial (BAP)*<sup>5</sup>, while being easy to compute [24]. While the following results will focus on the preliminaries for the later statements of this thesis, an in depth introduction to the properties of Chebyshev polynomials may for example be found in [26].

The following chapters 2.1.1, 2.1.2 and 2.1.3 are somewhat technical and will be abbreviated with some ideas and proofs only sketched. [26] and [29] build the foundation of these chapters and provide a deep insight into the topic.

#### 2.1.1 Expansion

The underlying theorem in Chebyshev approximation describes the representation of a continuous function as a Chebyshev series:

**Theorem 2.1.** Let  $f$  be Lipschitz continuous on  $[-1, 1]$ . Then  $f$  has a unique representation as a Chebyshev series:

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x).$$

The series is absolutely and uniformly convergent. The coefficients are given by:

$$a_k = \begin{cases} \frac{1}{\pi} \int_{-1}^1 \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx & \text{for } k = 0, \\ \frac{2}{\pi} \int_{-1}^1 \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx & \text{for } k \geq 1. \end{cases}$$

---

<sup>3</sup>The transcription of his name has been described as *one of the most well known data-retrieval nightmares in mathematical literature* [31]. This thesis will use the transcription adopted by the American Mathematical Society (see for example [2]).

<sup>4</sup>A measurement for the quality of an interpolation of a function for given nodes, cf. [28].

<sup>5</sup>For a given degree  $n$  the polynomial of  $\leq n^{\text{th}}$  degree with the smallest maximum deviation from the target function in the  $L^\infty$ -norm. [19]

To approximate a function  $f$  by a polynomial of degree  $n$  using Chebyshev polynomials the series is truncated after  $n$  terms:

$$f(x) \approx \check{f}(x) := \sum_{k=0}^n a_k T_k(x).$$

*Proof.* Omitted. See for example [29].  $\square$

Let  $K \in \mathbb{N}$ ,  $f : [-1, 1]^K \rightarrow \mathbb{R}$  holomorphic,  $\lambda$  be the measure on  $[-1, 1]$  with Lebesgue density  $(1 - x^2)^{-1/2}$  for  $x \in (-1, 1)$ . Then as shown in [29] for all  $y = (y_1, \dots, y_K) \in [-1, 1]^K$  the one-dimensional case (so  $K = 1$ )

$$f(y) = \sum_{k=0}^{\infty} \frac{2}{\pi} T_k(y) \int_{-1}^1 f(x) T_k(x) d\lambda(x)$$

can be expanded to

$$\begin{aligned} & f(y_1, \dots, y_K) \\ &= \sum_{k_1=0}^{\infty} \frac{2^{\mathbb{1}_{\mathbb{N}}(k_1)}}{\pi} T_{k_1}(y_1) \int_{-1}^1 f(x_1, y_2, \dots, y_K) T_{k_1}(x_1) d\lambda(x_1) \cdots \\ & \sum_{k_K}^{\infty} \frac{2^{\mathbb{1}_{\mathbb{N}}(k_K)}}{\pi} T_{k_K}(y_K) \int_{-1}^1 f(x_1, \dots, x_K) T_{k_K}(x_K) d\lambda(x_K) \cdots T_{k_1}(x_1) d\lambda(x_1) \\ &= \sum_{k \in \mathbb{N}_0^K} T_k(y) 2^{|k|_0} \pi^{-K} \int_{-1}^1 \cdots \int_{-1}^1 f(x) T_k(x) d\lambda(x_1) \cdots d\lambda(x_K) \\ &= \sum_{k \in \mathbb{N}_0^K} T_k(y) f_k. \end{aligned}$$

Now a bound for the Chebyshev coefficients  $f_k$  will be presented. Later this will be needed for a bound on the DNN error.

**Lemma 2.2.** Let  $K \in \mathbb{N}$ ,  $f : [-1, 1]^K \rightarrow \mathbb{R}$  a map that admits a holomorphic complex extension to  $\mathcal{E}_{\varrho} \subset \mathbb{C}^K$  with  $\varrho = (\rho, \dots, \rho) \in (1, \infty)^K$  for some  $\rho > 1$  as defined in Definition 1.6. Then for every  $k \in \mathbb{N}_0^K$ :

$$|f_k| \leq 2^{|k|_0} \max_{z \in \mathcal{E}_{\varrho}} |f(z)| \rho^{-k}, \quad \sum_{k \in \mathbb{N}_0^K} |f_k| \leq \left( \frac{2\rho}{\rho - 1} \right)^K \max_{z \in \mathcal{E}_{\varrho}} |f(z)|. \quad (1)$$

*Proof.* By [[29], Chapter 3] and [[26], Thm. 3.8] the coefficients of  $f$  satisfy

$$f_k = 2^{|k|_0} \pi^{-K} \int_{-1}^1 \cdots \int_{-1}^1 f(x) T_k(x) d\lambda(x_1) \cdots d\lambda(x_K) \quad (2)$$

$$\begin{aligned}
 &= 2^{|k|_0} (2\pi)^{-K} \int_{-\pi}^{\pi} \cdots \int_{-\pi}^{\pi} f(\cos(\theta)) \cos(k_1 \theta_1) d\theta_1 \cdots \cos(k_K \theta_K) d\theta_K \\
 &= 2^{|k|_0} (2\pi i)^{-K} \int_{\Gamma_1} \cdots \int_{\Gamma_1} \\
 &\quad f\left(\frac{z_1 + z_1^{-1}}{2}, \dots, \frac{z_K + z_K^{-1}}{2}\right) \left(\frac{z_1^{k_1} + z_1^{-k_1}}{2}\right) \frac{dz_1}{z_1} \cdots \left(\frac{z_K^{k_K} + z_K^{-k_K}}{2}\right) \frac{dz_K}{z_K} \\
 &= 2^{|k|_0} (2\pi i)^{-K} \int_{\Gamma_1} \cdots \int_{\Gamma_1} f\left(\frac{z_1 + z_1^{-1}}{2}, \dots, \frac{z_K + z_K^{-1}}{2}\right) z_1^{-k_1} \frac{dz_1}{z_1} \cdots z_K^{-k_K} \frac{dz_K}{z_K},
 \end{aligned}$$

where  $\Gamma_1$  is the unit circle in the complex plane as defined in Definition 1.6. The last equality is due to  $\frac{z+z^{-1}}{2}$  being invariant under the transformation  $z \mapsto z^{-1}$  for non-zero  $z \in \mathbb{C}$ . Now change the integration area from the unit circle to  $\Gamma_\rho$  (which is possible, since  $f$  is holomorphic):

$$\begin{aligned}
 |f_k| &= \left| 2^{|k|_0} (2\pi i)^{-K} \int_{\Gamma_\rho} \cdots \int_{\Gamma_\rho} f\left(\frac{z_1 + z_1^{-1}}{2}, \dots, \frac{z_K + z_K^{-1}}{2}\right) z^{-k} \frac{dz_1}{z_1} \cdots \frac{dz_K}{z_K} \right| \\
 &\leq 2^{|k|_0} \max_{z \in \mathcal{E}_\rho} |f(z)| (2\pi)^{-K} \left| \int_{\Gamma_\rho} \cdots \int_{\Gamma_\rho} z^{-k} \frac{dz_1}{z_1} \cdots \frac{dz_K}{z_K} \right| \\
 &\leq 2^{|k|_0} \max_{z \in \mathcal{E}_\rho} |f(z)| \rho^{-k}, \\
 \sum_{k \in \mathbb{N}_0^K} |f_k| &\leq 2^K \max_{z \in \mathcal{E}_\rho} |f(z)| \left(\frac{\rho}{\rho-1}\right)^K = \left(\frac{2\rho}{\rho-1}\right)^K \max_{z \in \mathcal{E}_\rho} |f(z)|.
 \end{aligned}$$

Here the first inequality derives from the fact that  $\mathcal{E}_\rho$  is a subset of the disc in the complex plane with radius  $\rho$  (so the disc with boundary  $\Gamma_\rho$ ).  $\square$

### 2.1.2 Interpolation

The next step towards Chebyshev interpolation is approximating the Chebyshev coefficients from chapter 2.1.1. For  $n \in \mathbb{N}$  and for  $k \in \{0, \dots, n\}^K$  let  $\hat{f}_{k;n}$  be the approximation of the Chebyshev coefficients one gets by applying an  $(n+1)^K$  point tensor product quadrature in the Clenshaw-Curtis<sup>6</sup> points  $x_j^n := \cos(j\pi/n)$

<sup>6</sup>A quadrature method based on Chebyshev polynomials, see [3].

for  $j \in \{0, \dots, n\}^K$  to the integrals in (2) with corresponding weights

$$\begin{aligned} w_j &:= \left(\frac{\pi}{2n}\right)^K \prod_{\ell=1}^K 2^{\mathbb{1}_{\{1, \dots, n-1\}}(j_\ell)}, \quad j \in \{0, \dots, n\}^K \\ \hat{f}_{k;n} &:= 2^{\mathbb{1}_{S(n)}(k)} \pi^{-K} \sum_{j \in \{0, \dots, n\}^K} w_j f(\cos(j\pi/n)) \cos(kj\pi/n) \\ &= 2^{\mathbb{1}_{S(n)}(k)} (2n)^{-K} \sum_{j \in \{0, \dots, 2n-1\}^K} f(\cos(j\pi/n)) \cos(kj\pi/n) \\ &= 2^{\mathbb{1}_{S(n)}(k)} \text{IFFT} \left( \left( f(x_j^n) \right)_{j \in \{0, \dots, 2n-1\}^K} \right)_k, \end{aligned}$$

where  $S(n) := \{1, \dots, n-1\}$  and  $kj = (k_1 j_1, \dots, k_K j_K) \in \mathbb{N}_0^K$  for  $k$  and  $j \in \{0, \dots, 2n-1\}^K$ . IFFT denotes the inverse fast Fourier transform given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{-i2\pi kn/N}, \quad n = 0, \dots, N-1$$

(cf. [25]). Since  $\cos((2n-j)\pi/n) = \cos(j\pi/n)$  for  $j = 1, \dots, n-1$ , it is sufficient to compute  $(f(x_j^n))_{j \in \{0, \dots, n\}^K}$  for  $(f(x_j^n))_{j \in \{0, \dots, 2n-1\}^K}$ . If  $k_j = n$  for some  $j = 1, \dots, K$ , the factor 2 in  $2^{\mathbb{1}_{S(n)}(k)}$  is dropped [[29], Chapter 4]. [[26], Thm. 3.13] implies that the approximation of  $f$  given by

$$\hat{p}_{f,n} := I_n^{\text{CC}}[f] := \sum_{k \in \{0, \dots, n\}^K} \hat{f}_{k;n} T_k \quad (3)$$

is the Lagrange interpolant in the Clenshaw-Curtis points. Furthermore, [[26], Thm. 1.2] and [[33], Thm. 4] prove that the Lebesgue constant in these points satisfies

$$\|I_n^{\text{CC}}\|_{L^\infty, L^\infty} := \|I_n^{\text{CC}}\|_{L^\infty([-1,1]^K), L^\infty([-1,1]^K)} \leq \left( \frac{2}{\pi} \log(n+1) + 1 \right)^K.$$

[[10], Chapter 2.2] shows a bound for all  $s \in \mathbb{N}$ :

$$\|I_n^{\text{CC}}\|_{W^{s,\infty}, W^{s,\infty}} \leq n^{2s} \|I_n^{\text{CC}}\|_{L^\infty, L^\infty}.$$

### 2.1.3 Interpolation Based on Approximate Function Values

Assuming the target function  $f$  is numerically approximated in the Clenshaw-Curtis points  $(x_j^n)_{j \in \{0, \dots, n\}^K}$  and assuming this approximation might be corrupted, call this approximation  $\check{f}$ . This shall be used for evaluating the function, for example in chapter 2.1.2. The coefficients and other values from chapter 2.1.2 are then

adapted and  $\hat{f}_{k;n}$  is further approximated. For all  $k \in \{0, \dots, n\}^K$  the previous results are updated to

$$\begin{aligned} \check{f}_{k;n} &:= 2^{\mathbb{1}_{S(n)}(k)} \pi^{-K} \sum_{j \in \{0, \dots, n\}^K} w_j \check{f}(\cos(j\pi/n)) \cos(kj\pi/n) \\ &= 2^{\mathbb{1}_{S(n)}(k)} (2n)^{-K} \sum_{j \in \{0, \dots, 2n-1\}^K} \check{f}(\cos(j\pi/n)) \cos(kj\pi/n) \\ &= 2^{\mathbb{1}_{S(n)}(k)} \text{IFFT} \left( \left( \check{f}(x_j^n) \right)_{j \in \{0, \dots, 2n-1\}^K} \right)_k, \end{aligned} \quad (4)$$

and update (3) to

$$\check{p}_{f,n} := \sum_{k \in \{0, \dots, n\}^K} \check{f}_{k;n} T_k = I_n^{\text{CC}}[\check{f}]. \quad (5)$$

Now error bounds (in different norms) for this can be postulated:

**Lemma 2.3.** Let  $K \in \mathbb{N}$ ,  $f : [-1, 1]^K \rightarrow \mathbb{R}$  a map which admits a holomorphic complex extension to the isotropic Bernstein polyellipse  $\mathcal{E}_\varrho$  with  $\varrho = (\rho, \dots, \rho) \in (1, \infty)^K$  for some  $\rho > 1$ . Assume that an approximation  $\check{f}$  of  $f$  is available and an upper bound on  $\|f - \check{f}\|_{L^\infty([-1, 1]^K)}$  exists. Then, for every  $\rho' \in (1, \rho)$  and every  $s \in \mathbb{N}$ , there exists a constant  $C''(s, \rho, \rho') > 0$  such that for all  $n \in \mathbb{N}$ :

$$\begin{aligned} &\|f - \check{p}_{f,n}\|_{L^\infty([-1, 1]^K)} \\ &\leq \left(1 + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty}\right) K \left(\frac{2\rho}{\rho - 1}\right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \rho^{-n-1} \\ &\quad + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1, 1]^K)}, \\ &\|f - \check{p}_{f,n}\|_{W^{s, \infty}([-1, 1]^K)} \\ &\leq \left(1 + \|I_n^{\text{CC}}\|_{W^{s, \infty}, W^{s, \infty}}\right) K \left(\frac{2C'\rho'}{\rho' - 1}\right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \rho'^{-n-1} \\ &\quad + \|I_n^{\text{CC}}\|_{L^\infty, W^{s, \infty}} \|f - \check{f}\|_{L^\infty([-1, 1]^K)}. \end{aligned}$$

*Proof.* For all  $q \in \mathbb{Q}_n$  use  $q = I_n^{\text{CC}}[q]$ . Then:

$$\begin{aligned} &\|f - \check{p}_{f,n}\|_{L^\infty([-1, 1]^K)} \\ &\leq \|f - q\|_{L^\infty([-1, 1]^K)} + \|q - \hat{p}_{f,n}\|_{L^\infty([-1, 1]^K)} + \|\hat{p}_{f,n} - \check{p}_{f,n}\|_{L^\infty([-1, 1]^K)} \\ &= \|f - q\|_{L^\infty([-1, 1]^K)} + \|I_n^{\text{CC}}[q - f]\|_{L^\infty([-1, 1]^K)} + \|I_n^{\text{CC}}[f - \check{f}]\|_{L^\infty([-1, 1]^K)} \\ &\leq \|f - q\|_{L^\infty([-1, 1]^K)} + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - q\|_{L^\infty([-1, 1]^K)} \\ &\quad + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1, 1]^K)}. \end{aligned}$$

Analogously:

$$\begin{aligned} & \|f - \check{p}_{f,n}\|_{W^{s,\infty}([-1,1]^K)} \\ & \leq \|f - q\|_{W^{s,\infty}([-1,1]^K)} + \|I_n^{\text{CC}}\|_{W^{s,\infty}, W^{s,\infty}} \|f - q\|_{W^{s,\infty}([-1,1]^K)} \\ & + \|I_n^{\text{CC}}\|_{L^\infty, W^{s,\infty}} \|f - \check{f}\|_{L^\infty([-1,1]^K)}. \end{aligned}$$

If  $f$  is holomorphic on the whole polydisk spanned by  $\mathcal{E}'_\varrho$  for  $\varrho' := (\rho', \dots, \rho') \in (1, \infty)^K$ , namely  $B_{\varrho'} := \times_{j=1}^K \{z_j \in \mathbb{C} : |z_j| \leq \rho'\}$ , the claim follows easily by taking  $q \in \mathbb{Q}_n$  to be the Taylor approximation of  $f$  in 0 and using the error bounds of the Taylor approximation (see [[11], Thm. 5.1] or [[10], Lemma 2.2]).

If this condition does not hold, take  $q$  as  $q := \sum_{k \in \{0, \dots, n\}^K} f_k T_k$ . This can be seen as a reduction of the Chebyshev expansion and now an error bound can be found using (1) on the Chebyshev coefficients. Write:  $\Lambda_n^c := \mathbb{N}_0^K \setminus \{0, \dots, n\}^K$  and  $k^{(j)} := (0, \dots, 0, \underbrace{n+1}_{k_j^{(j)}}, 0, \dots, 0) \in \mathbb{N}_0^K$  for  $j = 1, \dots, K$ . Since

$$\Lambda_n^c = \bigcup_{j=1}^K \{k^{(j)} + k : k \in \mathbb{N}_0^K\}$$

and  $\|T_k\|_{L^\infty([-1,1]^K)} = 1$  for all  $k \in \mathbb{N}_0^K$ , it follows that:

$$\begin{aligned} \|f - q\|_{L^\infty([-1,1]^K)} & \leq \sum_{\Lambda_n^c} |f_k| \|T_k\|_{L^\infty([-1,1]^K)} \leq \sum_{j=1}^K \sum_{k \in \mathbb{N}_0^K} |f_{k^{(j)}+k}| \\ & \leq \sum_{j=1}^K \sum_{k \in \mathbb{N}_0^K} 2^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \varrho^{-k^{(j)}-k} \\ & \leq K \left( \frac{2\rho}{\rho-1} \right)^K \max_{z \in \mathcal{E}_\rho} |f(z)| \rho^{-n-1}. \end{aligned}$$

In combination with the first part of the proof, the first claim now follows:

$$\begin{aligned} & \|f - \check{p}_{f,n}\|_{L^\infty([-1,1]^K)} \\ & \leq \|f - q\|_{L^\infty([-1,1]^K)} + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - q\|_{L^\infty([-1,1]^K)} \\ & + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\ & \leq \left(1 + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty}\right) K \left( \frac{2\rho}{\rho-1} \right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \rho^{-n-1} \\ & + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \end{aligned}$$

Now for the second claim note that for all  $\rho' \in (1, \rho)$  there exists  $C'(s, \rho, \rho') > 0$  such that  $\rho^{-k} k^{2s} \leq C' \rho'^{-k}$  for all  $k \in \mathbb{N}_0$ . In [[10], Lemma 2.2] it is shown that

$$\|T_{\mathbf{k}}\|_{W^{s,\infty}([-1,1]^K)} \leq \prod_{\ell=1}^K \max \{1, k_{\ell}^{2s}\}.$$

So for all  $\rho' \in (1, \rho)$ :

$$\begin{aligned} & \|f - q\|_{W^{s,\infty}([-1,1]^K)} \\ & \leq \sum_{\Lambda_n^c} |f_k| \|T_k\|_{W^{s,\infty}([-1,1]^K)} \\ & \leq \sum_{j=1}^K \sum_{k \in \mathbb{N}_0^K} |f_{k^{(j)}+k}| \prod_{\ell=1}^K \max \{1, (k^{(j)} + k)_{\ell}^{2s}\} \\ & \leq \sum_{j=1}^K \sum_{k \in \mathbb{N}_0^K} 2^K \max_{z \in \mathcal{E}_{\rho}} |f(z)| \rho^{-k^{(j)}-k} \prod_{\ell=1}^K \max \{1, (k^{(j)} + k)_{\ell}^{2s}\} \\ & \leq K 2^K \max_{z \in \mathcal{E}_{\rho}} |f(z)| \left( \sum_{k=0}^{\infty} \max \{1, k^{2s}\} \rho^{-k} \right)^{K-1} \left( \sum_{k=n+1}^{\infty} \max \{1, k^{2s}\} \rho^{-k} \right) \\ & \leq K 2^K \max_{z \in \mathcal{E}_{\rho}} |f(z)| \left( \sum_{k=0}^{\infty} C' \rho'^{-k} \right)^{K-1} \left( \sum_{k=n+1}^{\infty} C' \rho'^{-k} \right) \\ & \leq K \left( \frac{2C' \rho'}{\rho' - 1} \right)^K \max_{z \in \mathcal{E}_{\rho}} |f(z)| \rho'^{-n-1}. \end{aligned}$$

In combination with the first part of the proof, the second claim now follows:

$$\begin{aligned} & \|f - \check{p}_{f,n}\|_{W^{s,\infty}([-1,1]^K)} \\ & \leq \|f - q\|_{W^{s,\infty}([-1,1]^K)} + \|I_n^{\text{CC}}\|_{W^{s,\infty}, W^{s,\infty}} \|f - q\|_{W^{s,\infty}([-1,1]^K)} \\ & \quad + \|I_n^{\text{CC}}\|_{L^{\infty}, W^{s,\infty}} \|f - \check{f}\|_{L^{\infty}([-1,1]^K)} \\ & \leq \left(1 + \|I_n^{\text{CC}}\|_{W^{s,\infty}, W^{s,\infty}}\right) K \left(\frac{2C' \rho'}{\rho' - 1}\right)^K \max_{z \in \mathcal{E}_{\rho}} |f(z)| \rho'^{-n-1} \\ & \quad + \|I_n^{\text{CC}}\|_{L^{\infty}, W^{s,\infty}} \|f - \check{f}\|_{L^{\infty}([-1,1]^K)}. \end{aligned}$$

□

This error bound finally leads to the error estimate for approximated Chebyshev coefficients:

**Corollary 2.4.** Let the assumptions of Lemma 2.3 be true. Then, for all  $n \in \mathbb{N}$

$$\begin{aligned} & \sum_{k \in \{0, \dots, n\}^K} \left| \check{f}_{k;n} \right| \\ & \leq C(K, \rho) \max_{z \in \mathcal{E}_\varrho} |f(z)| + (n+1)^K \pi^K \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)}. \end{aligned}$$

*Proof.*

$$\begin{aligned} & \sum_{k \in \{0, \dots, n\}^K} \left| \check{f}_{k;n} \right| \\ & \leq \sum_{k \in \mathbb{N}_0^K} |f_k| + \sum_{k \in \{0, \dots, n\}^K} \left| f_k - \check{f}_{k;n} \right| \\ & \leq \left( \frac{2\rho}{\rho-1} \right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| + (n+1)^K \pi^K \|f - \check{p}_{f,n}\|_{L^\infty([-1,1]^K)} \\ & \leq \left( \frac{2\rho}{\rho-1} \right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \left[ 1 + K(n+1)^K \pi^K \left( 1 + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \right) \rho^{-n-1} \right] \\ & \quad + (n+1)^K \pi^K \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)}. \end{aligned}$$

The second inequality results from

$$\begin{aligned} & \left| f_k - \check{f}_{k;n} \right| \\ & \leq \left| \int_{-1}^1 \cdots \int_{-1}^1 (f - \check{p}_{f,n}) T_k \, d\lambda(x_1) \cdots d\lambda(x_K) \right| \\ & \leq \|f - \check{p}_{f,n}\|_{L^\infty([-1,1]^K)} \|T_k\|_{L^\infty([-1,1]^K)} \left| \int_{-1}^1 \cdots \int_{-1}^1 d\lambda(x_1) \cdots d\lambda(x_K) \right| \\ & = \pi^K \|f - \check{p}_{f,n}\|_{L^\infty([-1,1]^K)}. \end{aligned}$$

□

After having established some properties of Chebyshev approximation, the main results can be presented. Firstly approximation of Chebyshev polynomials will be discussed by presenting the approximation of Chebyshev polynomials of the first kind and how this concept may be adopted to achieve approximation of tensor product Chebyshev polynomials. This will then be combined with the results of the previous chapters.



## 2.2 ReLU DNN Approximation of Chebyshev Polynomials

### 2.2.1 Multiplication by DNNs

The following auxiliary lemma can be found in [27] as Proposition 3.1. It shows the existence of ReLU DNNs that approximate the multiplication of two numbers in an interval while their size and depth increase merely logarithmically in the accuracy.

**Lemma 2.5.** For any  $\delta \in (0, 1)$  and  $M \geq 1$  there exists a ReLU DNN  $\tilde{\times}_{\delta, M} : [-M, M]^2 \rightarrow \mathbb{R}$  such that

$$\sup_{|a|, |b| \leq M} |ab - \tilde{\times}_{\delta, M}(a, b)| \leq \delta,$$

$$\text{ess sup}_{|a|, |b| \leq M} \max \left\{ \left| b - \frac{\partial}{\partial a} \tilde{\times}_{\delta, M}(a, b) \right|, \left| a - \frac{\partial}{\partial b} \tilde{\times}_{\delta, M}(a, b) \right| \right\} \leq \delta,$$

where  $\frac{\partial}{\partial a} \tilde{\times}_{\delta, M}(a, b)$  and  $\frac{\partial}{\partial b} \tilde{\times}_{\delta, M}(a, b)$  denote weak derivatives.

There exists a constant  $C > 0$  independent of  $\delta$  and  $M$  such that  $\text{size}_{\text{in}}(\tilde{\times}_{\delta, M}) \leq C$ ,  $\text{size}_{\text{out}}(\tilde{\times}_{\delta, M}) \leq C$ ,

$$\text{depth}(\tilde{\times}_{\delta, M}) \leq C(1 + \log_2(M/\delta)), \quad \text{size}(\tilde{\times}_{\delta, M}) \leq C(1 + \log_2(M/\delta)).$$

Moreover, for every  $a \in [-M, M]$ , there exists a finite set  $\mathcal{N}_a \subseteq [-M, M]$  such that  $b \mapsto \tilde{\times}_{\delta, M}(a, b)$  is strongly differentiable at all  $b \in (-M, M) \setminus \mathcal{N}_a$ .

*Proof.* Existence and the bounds on size and depth will not be proven explicitly but can easily be deduced from the actual implementation of these DNNs, which will be discussed in chapter 3.

First define the *sawtooth function*  $g : [0, 1] \rightarrow [0, 1]$  as

$$g(x) = \begin{cases} 2x & \text{if } x < \frac{1}{2} \\ 2(1-x) & \text{if } x \geq \frac{1}{2} \end{cases}$$

and define the  $m$ -fold composition as  $g_m = \underbrace{g \circ \dots \circ g}_{m \text{ times}}$ . Define the continuous, piecewise linear spline interpolation of  $x^2$  on  $[0, 1]$  at the  $2^m + 1$  uniformly distributed nodes  $j2^{-m}$  for  $j = 0, \dots, 2^m \in \mathbb{N}_0$  as  $f_m : [0, 1] \rightarrow [0, 1]$  recursively with  $f_0(x) := x$  and

$$f_m(x) = f_{m-1}(x) - \frac{g_m(x)}{2^{2m}} \quad \forall m \geq 1.$$

It holds that the pointwise error is given by

$$\sup_{x \in [0, 1]} |x^2 - f_m(x)| = 2^{-2m-2}.$$

The fact that  $f_m$  actually is the piecewise linear interpolation of  $x^2$  and that the error holds can be verified in proposition 2 of [32]. For any  $M > 0$  and  $a, b \in [-M, M]$  it holds that

$$\begin{aligned} (a+b)^2 &= a^2 + 2ab + b^2 \\ \Leftrightarrow ab &= \frac{1}{2} ((a+b)^2 - a^2 - b^2) \\ \Leftrightarrow ab &= \frac{2M^2}{4M^2} (|a+b|^2 - |a|^2 - |b|^2) \\ \Leftrightarrow ab &= 2M^2 \left( \left( \frac{|a+b|}{2M} \right)^2 - \left( \frac{|a|}{2M} \right)^2 - \left( \frac{|b|}{2M} \right)^2 \right). \end{aligned}$$

The functions  $g$ ,  $g_m$  and  $f_m$  can be written as DNNs (again, see chapter 3 for the implementation). By defining

$$\tilde{\times}_{\delta, M}(a, b) := 2M^2 \left( f_m \left( \frac{|a+b|}{2M} \right) - f_m \left( \frac{|a|}{2M} \right) - f_m \left( \frac{|b|}{2M} \right) \right)$$

and recounting that the error for  $f_m$  can be made arbitrarily small by choosing a large enough  $m$ , the claim  $\sup_{|a|, |b| \leq M} |ab - \tilde{\times}_{\delta, M}(a, b)| \leq \delta$  for any  $\delta \in (0, 1)$  follows.

Now consider  $f'_m(x) := \frac{\partial}{\partial x} f_m(x)$ . Then

$$\begin{aligned} &\text{ess sup}_{x \in [0, 1]} |2x - f'_m(x)| \\ &= \sup_{j=0, \dots, 2^m-1} \sup_{x \in [x_j, x_{j+1}]} |2x - f'_m(x)| = \sup_{j=0, \dots, 2^m-1} \sup_{x \in [x_j, x_{j+1}]} \left| 2x - \frac{x_{j+1}^2 - x_j^2}{x_{j+1} - x_j} \right| \\ &= \sup_{j=0, \dots, 2^m-1} \sup_{x \in [x_j, x_{j+1}]} |2x - (x_{j+1} + x_j)| = \sup_{j=0, \dots, 2^m-1} |x_{j+1} - x_j| = 2^{-m}. \end{aligned}$$

For every  $a, b \in [0, M] \setminus \{2Mx_j : j = 0, \dots, 2^m\}$  such that  $a+b \notin \{2Mx_j : j = 0, \dots, 2^m\}$

$$\left| b - \frac{\partial}{\partial a} \tilde{\times}_{\delta, M}(a, b) \right| = M \left| \frac{2(a+b)}{2M} - \frac{2a}{2M} - f'_m \left( \frac{a+b}{2M} \right) + f'_m \left( \frac{a}{2M} \right) \right| \leq 2M2^{-m}.$$

Choose  $m \geq -\log_2(\delta/(2M))$  which gives  $\text{ess sup}_{0 \leq a, b \leq M} |b - \frac{\partial}{\partial a} \tilde{\times}_{\delta, M}(a, b)| \leq \delta$ . The other cases follow analogously.

Lastly the existence of  $\mathcal{N}_a$  follows from  $f_m$  being the piecewise linear spline interpolation of  $x^2$  in  $2^m + 1$  many points and from the definition of  $\tilde{\times}_{\delta, M}$ .  $\square$

This gives the foundation for ReLU DNNs that are capable of multiplying  $n$  different numbers with tameable increase in size and depth for higher accuracy. To achieve this the following result will be proven by constructing a binary tree of networks as given in Lemma 2.5.

**Proposition 2.6.** For any  $\delta \in (0, 1)$ ,  $n \in \mathbb{N}$  and  $M \geq 1$  there exists a ReLU DNN  $\tilde{\Pi}_{\delta, M}^n : [-M, M]^n \rightarrow \mathbb{R}$  such that

$$\sup_{(x_i)_{i=1}^n \in [-M, M]^n} \left| \prod_{j=1}^n x_j - \tilde{\Pi}_{\delta, M}^n(x_1, \dots, x_n) \right| \leq \delta, \quad (6)$$

$$\text{ess sup}_{(x_i)_{i=1}^n \in [-M, M]^n} \sup_{i=1, \dots, n} \left| \frac{\partial}{\partial x_i} \prod_{j=1}^n x_j - \frac{\partial}{\partial x_i} \tilde{\Pi}_{\delta, M}^n(x_1, \dots, x_n) \right| \leq \delta. \quad (7)$$

There exists a constant  $C$  independent of  $\delta \in (0, 1)$ ,  $n \in \mathbb{N}$  and  $M \geq 1$  such that

$$\text{size} \left( \tilde{\Pi}_{\delta, M}^n \right) \leq C (1 + n \log_2 (nM^n/\delta)), \quad (8)$$

$$\text{depth} \left( \tilde{\Pi}_{\delta, M}^n \right) \leq C (1 + \log_2(n) \log_2 (nM^n/\delta)). \quad (9)$$

*Proof.* Define  $\tilde{n} := \min \{2^k : k \in \mathbb{N}, 2^k \geq n\}$  and consider the product of  $\tilde{n}$  numbers  $x_1, \dots, x_{\tilde{n}} \in [-M, M]$ . In case  $n < \tilde{n}$ , let  $x_{n+1}, \dots, x_{\tilde{n}} := 1$  (this can be implemented by a bias in the first layer). By definition  $\tilde{n} < 2n$ , so the bounds size and depth in terms of  $\tilde{n}$  also hold in terms of  $n$ , possibly with a larger constant. It is enough to show the result for  $M = 1$ , since if the claim is true for  $M = 1$  then for any given  $\tilde{M} > 1$ , one can define

$$\tilde{\Pi}_{\delta, \tilde{M}}(x_1, \dots, x_n) := \tilde{M}^n \tilde{\Pi}_{\delta/\tilde{M}^n, 1}(x_1/\tilde{M}, \dots, x_n/\tilde{M}).$$

As can easily be seen this will indeed achieve the claim for all  $(x_i)_{i=1}^n \in [-\tilde{M}, \tilde{M}]^n$ . Therefore, w.l.o.g.  $M = 1$  throughout the rest of this proof.

By abuse of notation, for every even  $k \in \mathbb{N}$  and using the networks  $\tilde{\times}(\cdot, \cdot)$  given by Lemma 2.5, let a ( $k$ -dependent) mapping  $R = R^1$  be defined via

$$R(y_1, \dots, y_k) := (\tilde{\times}_{\delta/\tilde{n}^2, 2}(y_1, y_2), \dots, \tilde{\times}_{\delta/\tilde{n}^2, 2}(y_{k-1}, y_k)) \in \mathbb{R}^{k/2}.$$

For  $\ell \geq 2$  define recursively  $R^\ell := R \circ R^{\ell-1}$ . Hence,  $R^\ell$  can be considered as an recursive algorithm for pairwise multiplication of the entries of an even-dimensional vector and can be interpreted as a mapping from  $\mathbb{R}^{2^\ell} \rightarrow \mathbb{R}$ . Finally define  $\tilde{\Pi}_{\delta, 1} : [-1, 1]^n \rightarrow \mathbb{R}$  via

$$\tilde{\Pi}_{\delta, 1}(x_1, \dots, x_n) := R^{\log_2(\tilde{n})}(x_1, \dots, x_{\tilde{n}}).$$

For the error bounds it first has to be shown that for  $\ell \in \{1, \dots, \log_2(\tilde{n})\}$  and for all  $x_1, \dots, x_{2^\ell} \in [-1, 1]$

$$\left| \prod_{j=1}^{2^\ell} x_j - R^\ell(x_1, \dots, x_{2^\ell}) \right| \leq \delta \frac{2^{2^\ell}}{\tilde{n}^2}. \quad (10)$$

For  $\ell = 1$  it holds that  $R(x_1, x_2) = \tilde{\times}_{\delta/\tilde{n}^2, 2}(x_1, x_2)$  and by Lemma 2.5 the accuracy is given by  $\frac{\delta}{\tilde{n}^2}$ , hence (10) follows. Now assume for induction that for an arbitrary  $\ell \in \{2, \dots, \log_2(\tilde{n})\}$  equation (10) holds for  $\ell - 1$ . Since  $\left| \prod_{j=1}^{2^{\ell-1}} x_j \right| \leq 1$  (by assumption  $x_1, \dots, x_{2^\ell} \in [-1, 1]$ ) and  $\frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta < 1$  (since  $\ell \leq \log_2(\tilde{n})$ , so  $\frac{2^{2(\ell-1)}}{\tilde{n}^2} < 1$  and since  $\delta < 1$ ), it follows that  $|R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}})| < 2$ , hence  $R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}) \in [-2, 2]$  and hence may be used as input of  $\tilde{\times}_{\delta/\tilde{n}^2, 2}$ . This leads to

$$\begin{aligned} & \left| \prod_{j=1}^{2^\ell} x_j - R^\ell(x_1, \dots, x_{2^\ell}) \right| \\ & \leq \left| \prod_{j=1}^{2^{\ell-1}} x_j - R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}) \right| \cdot \left| \prod_{j=2^{\ell-1}+1}^{2^\ell} x_j \right| \\ & \quad + |R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}})| \cdot \left| \prod_{j=2^{\ell-1}+1}^{2^\ell} x_j - R^{\ell-1}(x_{2^{\ell-1}+1}, \dots, x_{2^\ell}) \right| \\ & \quad + |R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}) R^{\ell-1}(x_{2^{\ell-1}+1}, \dots, x_{2^\ell}) \\ & \quad - \tilde{\times}_{\delta/\tilde{n}^2, 2}(R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}), R^{\ell-1}(x_{2^{\ell-1}+1}, \dots, x_{2^\ell}))| \\ & \leq \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta + \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta \underbrace{\left( 1 + \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta \right)}_{\leq 2} + \frac{1}{\tilde{n}^2} \delta \\ & \leq \frac{2^{2(\ell-1)} + 2 \cdot 2^{2(\ell-1)} + 1}{\tilde{n}^2} \delta \leq \frac{2^{2^\ell}}{\tilde{n}^2} \delta. \end{aligned}$$

This finishes the induction to show (10) and choosing  $\ell = \log_2(\tilde{n})$  yields claim (6). Now show claim (7). Due to the symmetry of the  $\tilde{\times}_{\delta/\tilde{n}^2, 2}$ -networks, one can consider only the derivative with respect to  $x_1$  w.l.o.g.. For  $\ell \in \{1, \dots, \log_2(\tilde{n})\}$  show again by induction that for almost every  $(x_i)_{i=1}^{2^\ell} \in [-1, 1]^{2^\ell}$ :

$$\left| \frac{\partial}{\partial x_1} \prod_{j=1}^{2^\ell} x_j - \frac{\partial}{\partial x_1} R^\ell(x_1, \dots, x_{2^\ell}) \right| \leq \delta \frac{2^{2^\ell}}{\tilde{n}^2}. \quad (11)$$

As before,  $R(x_1, x_2) = \tilde{\times}_{\delta/\tilde{n}^2, 2}(x_1, x_2)$  and for  $\ell = 1$  the claim follows from Lemma 2.5. For  $\ell > 1$ , under the assumption that (11) holds for  $\ell - 1$ , it follows that

$$\begin{aligned}
 & \left| \frac{\partial}{\partial x_1} \prod_{j=1}^{2^\ell} x_j - \frac{\partial}{\partial x_1} R^\ell(x_1, \dots, x_{2^\ell}) \right| \\
 & \leq \left| \prod_{j=2^{\ell-1}+1}^{2^\ell} x_j \right| \cdot \left| \frac{\partial}{\partial x_1} \prod_{j=1}^{2^{\ell-1}} x_j - \frac{\partial}{\partial x_1} R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}) \right| \\
 & + \left| \prod_{j=2^{\ell-1}+1}^{2^\ell} x_j - R^{\ell-1}(x_{2^{\ell-1}+1}, \dots, x_{2^\ell}) \right| \cdot \left| \frac{\partial}{\partial x_1} R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}) \right| \\
 & + \left| R^{\ell-1}(x_{2^{\ell-1}+1}, \dots, x_{2^\ell}) \right. \\
 & \quad \left. - \left( \frac{\partial}{\partial a} \tilde{\times}_{\delta/\tilde{n}^2, 2} \right) (R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}), R^{\ell-1}(x_{2^{\ell-1}+1}, \dots, x_{2^\ell})) \right. \\
 & \quad \left. \cdot \left| \frac{\partial}{\partial x_1} R^{\ell-1}(x_1, \dots, x_{2^{\ell-1}}) \right| \right| \\
 & \leq \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta + \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta \underbrace{\left( 1 + \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta \right)}_{\leq 2} + \frac{1}{\tilde{n}^2} \delta \underbrace{\left( 1 + \frac{2^{2(\ell-1)}}{\tilde{n}^2} \delta \right)}_{\leq 2} \\
 & \leq \frac{2^{2(\ell-1)} + 2 \cdot 2^{2(\ell-1)} + 2}{\tilde{n}^2} \delta \leq \frac{2^{2\ell}}{\tilde{n}^2} \delta,
 \end{aligned}$$

where  $\frac{\partial}{\partial a} \tilde{\times}_{\delta/\tilde{n}^2, 2}$  denotes the weak derivative of  $\tilde{\times}_{\delta/\tilde{n}^2, 2} : [-2, 2] \times [-2, 2] \rightarrow \mathbb{R}$  w.r.t. its first argument as in Lemma 2.5. This finishes the induction for claim (11) and setting  $\ell = \log_2(\tilde{n})$  yields claim (7).

Finally, the number of repetitions of applying  $R$  (i.e. the number of binary tree layers) is bounded by  $\mathcal{O}(\log_2(\tilde{n}))$ . With the bound on the network depth from Lemma 2.5, for  $M = 1$  the claim (9) follows.

Since by 1.1 it holds for two DNNs  $\Phi_1, \Phi_2$  that  $\text{size}(\Phi_1 \circ \Phi_2) \leq 2 \text{size}(\Phi_1) + 2 \text{size}(\Phi_2)$  and obviously by definition  $\text{size}(\Phi) \geq \text{size}_{\text{in}}(\Phi)$  and  $\text{size}(\Phi) \geq \text{size}_{\text{out}}(\Phi)$  for any DNN  $\Phi$  use

$$\text{size}(\Phi_1 \circ \Phi_2) \leq \text{size}(\Phi_1) + \text{size}_{\text{in}}(\Phi_1) + \text{size}_{\text{out}}(\Phi_2) + \text{size}(\Phi_2)$$

and the bounds from Lemma 2.5. It follows  $(2^{\log_2(\tilde{n})-\ell})$  being the number of product

networks in binary tree layer  $\ell$  )

$$\begin{aligned}
 \text{size} \left( \tilde{\prod}_{\delta,1} \right) &\leq \sum_{\ell=1}^{\log_2(\tilde{n})} 2^{\log_2(\tilde{n})-\ell} \left( \text{size in } (\tilde{x}_{\delta/\tilde{n}^2,2}) + \text{size} (\tilde{x}_{\delta/\tilde{n}^2,2}) \right. \\
 &\quad \left. + \text{size}_{\text{out}} (\tilde{x}_{\delta/\tilde{n}^2,2}) \right) \\
 &\leq \sum_{\ell=1}^{\log_2(\tilde{n})} 2^{\log_2(\tilde{n})-\ell} (C + C(1 + \log_2(2\tilde{n}^2/\delta)) + C) \\
 &\leq (\tilde{n} - 1)C(1 + \log_2(\tilde{n}/\delta)) \leq C(1 + n \log_2(n/\delta)).
 \end{aligned}$$

□

### 2.2.2 Approximation of Chebyshev Polynomials of the First Kind

After having established DNNs that are able to emulate multiplications, the following chapter will introduce DNNs that approximate univariate Chebyshev polynomials  $T_n(x)$  as defined in definition 1.3. First existence of these DNNs is shown by construction. This construction process is described in the Appendix of [10], while the error bound and the bounds on depth and size have been proven by the author of this thesis<sup>7</sup>.

**Lemma 2.7.** There exists  $C > 0$  such that for all  $n \in \mathbb{N}$  and  $\delta \in (0, 1)$  there exist ReLU DNNs  $\left\{ \Phi_{\delta}^{\text{Cheb},n} \right\}_{\delta \in (0,1)}$  with input dimension one and output dimension  $n$  which satisfy

$$\left\| T_{\ell} - \left( \Phi_{\delta}^{\text{Cheb},n} \right)_{\ell} \right\|_{W^{1,\infty}([-1,1])} \leq \delta, \quad \ell = 1, \dots, n$$

and

$$\begin{aligned}
 \text{depth} \left( \Phi_{\delta}^{\text{Cheb},n} \right) &\leq C(1 + \log_2(n)) \log_2(1/\delta) + C(1 + \log_2(n))^3, \\
 \text{size} \left( \Phi_{\delta}^{\text{Cheb},n} \right) &\leq Cn \log_2(1/\delta) + Cn(1 + \log_2(n)).
 \end{aligned}$$

*Proof.*  $\left\{ \Phi_{\delta}^{\text{Cheb},n} \right\}_{\delta \in (0,1)}$  can be explicitly constructed:

Let  $I := (-1, 1)$ . As a first step construct for all  $k \in \mathbb{N}$  the DNNs  $\left\{ \Psi_{\delta}^k \right\}_{\delta \in (0,1)}$ , that have input dimension one and output dimension  $2^{k-1} + 2$  and that fulfill

---

<sup>7</sup>While the error bounds and the bounds on depth and size are stated in [10], they are not proven there, but instead the proof is announced to be published in the near future. Further information could not be provided by the authors prior to publication.

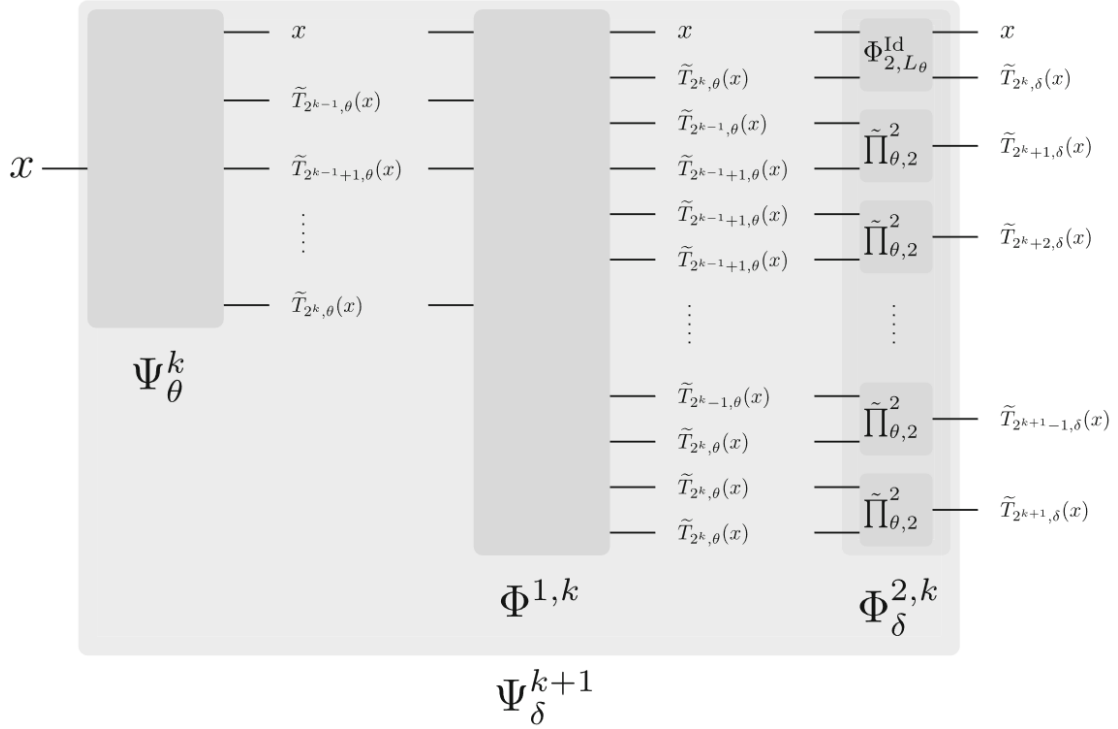


Figure 1: Sketch of  $\Psi_\delta^{k+1}$  for some  $k \in \mathbb{N}$  and  $\delta \in (0, 1)$ , inductively constructed from  $\Psi_\theta^k$  with  $\theta = 2^{-2k-4}\delta$ . The subnetwork  $\Phi^{1,k}$  realizes a linear map, correctly coupling the output of  $\Psi_\theta^k$  to the input of  $\Phi_\delta^{2,k}$ . The subnetwork  $\Phi_\delta^{2,k}$  acts as the identity on the first two inputs, and as an approximate multiplication from Proposition 2.6 on pairs of the remaining inputs. Output are the input  $x$  and approximations of Chebyshev polynomials of degree  $2^k, \dots, 2^{k+1}$ , with accuracy  $\delta$ . Source: [10]

the following properties: for  $\ell \in \{2^{k-1}, \dots, 2^k\}$  denote all but the first output components by  $\tilde{T}_{\ell,\delta} := (\Psi_\delta^k)_{2+\ell-2^{k-1}}$ . The first output component is  $x$  for  $x \in I$ , i.e.

$$\Psi_\delta^k(x) = \left( x, \tilde{T}_{2^{k-1},\delta}(x), \dots, \tilde{T}_{2^k,\delta}(x) \right), \quad x \in I.$$

Furthermore it holds that

$$\left\| T_\ell(x) - \tilde{T}_{\ell,\delta}(x) \right\|_{W^{1,\infty}(I)} \leq \delta, \quad \ell \in \{2^{k-1}, \dots, 2^k\}. \quad (12)$$

The DNN  $\Psi_\delta^k(x)$  is constructed inductively. For  $k = 1$  let  $\delta \in (0, 1)$  and set  $L_1 := \text{depth}(\tilde{\Pi}_{\delta/4,1}^2)$  as the depth of a multiplication network as constructed in proposition 2.6 for two inputs, error bound  $\delta/4$  and interval border  $M = 1$ . Let  $A_i, b_i$  for  $i = 1, \dots, L_1 + 1$  be the weights and biases of  $\tilde{\Pi}_{\delta/4,1}^2$ . Furthermore let  $A := [1, 1]^\top \in \mathbb{R}^{2 \times 1}$  and  $b := -1 \in \mathbb{R}$ . Define  $\Phi$  as the DNN with weights  $A_1 A, A_2, \dots, A_{L_1}, 2A_{L_1+1}$  and the biases  $b_1, \dots, b_{L_1}, b_{L_1+1} + b$ . With the parallelization defined in definition 1.2 now define:

$$\Psi_\delta^1 := (\Phi_{1,L_1}^{\text{Id}}, \Phi_{1,L_1}^{\text{Id}}, \Phi),$$

where  $\Phi_{1,L_1}^{\text{Id}}$  is the identity network with depth  $L_1$  as defined in definition 1.2.

It holds that  $(\Psi_\delta^1(x))_1 = \Phi_{1,L_1}^{\text{Id}}(x) = x$ ,  $\tilde{T}_{1,\delta}(x) := (\Psi_\delta^1(x))_2 = \Phi_{1,L_1}^{\text{Id}}(x) = x = T_1(x)$  and  $\tilde{T}_{2,\delta}(x) := (\Psi_\delta^1(x))_3 = 2 \left( \tilde{\Pi}_{\delta/4,1}^2(x, x) \right) - 1$ . With the definition of  $T_0, T_1$  and  $T_{m+n}$  from definition 1.3, the first and second component clearly emulate  $T_0$  and  $T_1$  with an error of 0 (since the identity DNN does not produce an error) while the third component emulates  $T_2$ . For the error recall Proposition 2.6 and consider the derivatives of  $T_2$  and  $\tilde{T}_{2,\delta}$ :

$$\begin{aligned} & \left\| 4x - 2 \left[ D \tilde{\Pi}_{\delta/4,1}^2(x, x) \right]_1 - 2 \left[ D \tilde{\Pi}_{\delta/4,1}^2(x, x) \right]_2 \right\|_{L^\infty(I)} \\ &= \left\| 2 \left( x - \left[ D \tilde{\Pi}_{\delta/4,1}^2(x, x) \right]_1 \right) + 2 \left( x - \left[ D \tilde{\Pi}_{\delta/4,1}^2(x, x) \right]_2 \right) \right\|_{L^\infty(I)} \\ &\leq 2 \frac{\delta}{4} + 2 \frac{\delta}{4} = \delta, \end{aligned}$$

where  $D$  is the Jacobian. Now it follows from Poincaré's inequality (cf. [[16], Thm. 1.1]), that

$$\left\| T_2(x) - \tilde{T}_{2,\delta}(x) \right\|_{W^{1,\infty}(I)} \leq \delta.$$

The claim (12) therefore holds for  $k = 1$ .



Now for all  $\delta \in (0, 1)$  and  $k \in \mathbb{N}$  define  $\theta := 2^{-2k-4}$ . For the purpose of induction assume that a DNN  $\Psi_\theta^k$  exists that fulfills the bound (12) with  $\theta$  instead of  $\delta$ . Define the DNN  $\Phi^{1,k}$  with depth 0 by its weight matrix  $A^{1,k} \in \mathbb{R}^{(2^{k+1}+2) \times (2^{k-1}+2)}$  with

$$(A^{1,k})_{m,i} = \begin{cases} 1 & \text{if } m = 1, i = 1 \\ 1 & \text{if } m = 2, i = 2^{k-1} + 2 \\ 1 & \text{if } m \in \{3, \dots, 2^{k+1} + 2\}, i = \lceil \frac{m+5}{4} \rceil, \\ 0 & \text{else,} \end{cases}$$

and its bias vector  $b^{1,k} = 0 \in \mathbb{R}^{2^{k+1}+2}$ . This DNN realizes

$$\mathbb{R}^{2^{k-1}+2} \rightarrow \mathbb{R}^{2^{k+1}+2} : (z_1, \dots, z_{2^{k-1}+2}) \mapsto (z_1, z_{2^{k-1}+2}, z_2, z_3, z_3, z_3, z_3, z_4, z_4, z_4, z_4, z_5, \dots, z_{2^{k-1}+1}, z_{2^{k-1}+2}, z_{2^{k-1}+2}, z_{2^{k-1}+2}).$$

Next, define the DNN  $\Phi'$  with depth 0 by its weight matrix  $A^{2,k} \in \mathbb{R}^{(2^k+2) \times (2^k+2)}$  and its bias  $b^{2,k} \in \mathbb{R}^{2^k+2}$  with

$$(A^{2,k})_{m,i} := \begin{cases} 1 & \text{if } m = i \leq 2 \\ 2 & \text{if } m = i \geq 3, \\ -1 & \text{if } m \geq 3 \text{ is odd, } i = 1, \\ 0 & \text{else,} \end{cases} \quad (b^{2,k})_m = \begin{cases} -1 & \text{if } m \geq 3 \text{ is even,} \\ 0 & \text{else.} \end{cases}$$

let  $L_\theta := \text{depth}(\tilde{\Pi}_{\theta,2}^2)$ . Define

$$\Phi_\delta^{2,k} := \Phi' \circ \left( \Phi_{2,L_\theta}^{\text{Id}}, \underbrace{\tilde{\Pi}_{\theta,2}^2, \dots, \tilde{\Pi}_{\theta,2}^2}_{2^k \text{ times}} \right)_d.$$

Note that, for all  $\ell \in \{2^{k-1}, \dots, 2^k\}$ , it holds that:

$$\left\| \tilde{T}_{\ell,\theta}(x) \right\|_{L^\infty(I)} \leq \|T_\ell(x)\|_{L^\infty(I)} + \left\| T_\ell(x) - \tilde{T}_{\ell,\theta}(x) \right\|_{W^{1,\infty}(I)} \leq 1 + \theta < 2,$$

hence  $\tilde{T}_{\ell,\theta}(x)$  may be used as an argument in  $\tilde{\Pi}_{\theta,2}^2$ . With this then define:

$$\Psi_\delta^{k+1} := \Phi_\delta^{2,k} \circ \Phi^{1,k} \circ \Psi_\theta^k.$$

For  $x \in I$  the DNN  $\Psi_\delta^{k+1}$  realizes:

$$\begin{aligned} (\Psi_\delta^{k+1}(x))_1 &= x, \\ (\Psi_\delta^{k+1}(x))_2 &= \tilde{T}_{2^k,\theta}(x), \end{aligned}$$

and for  $\ell \in \{2^k + 1, \dots, 2^{k+1}\}$ :

$$\tilde{T}_{\ell,\delta}(x) := (\Psi_\delta^{k+1}(x))_{\ell+2-2^k} = 2 \prod_{\theta,2}^2 \left( \tilde{T}_{\lceil \ell/2 \rceil, \theta}(x), \tilde{T}_{\lfloor \ell/2 \rfloor, \theta}(x) \right) - x^{\lceil \ell/2 \rceil - \lfloor \ell/2 \rfloor},$$

where  $x^{\lceil \ell/2 \rceil - \lfloor \ell/2 \rfloor} = x = T_1(x)$  for odd  $\ell$  and  $x^{\lceil \ell/2 \rceil - \lfloor \ell/2 \rfloor} = 1 = T_0(x)$  for even  $\ell$ .

Assuming the error bound holds for  $k$  and assuming  $\ell$  is even, first note that for  $m \in \{\lceil \ell/2 \rceil, \lfloor \ell/2 \rfloor\}$ :

$$\begin{aligned} \left\| \tilde{T}_{m,\theta}(x) \right\|_{L^\infty(I)} &\leq 1 + \theta \leq 2 \\ \left\| \frac{d}{dx} \tilde{T}_{m,\theta}(x) \right\|_{L^\infty(I)} &\leq \left\| \frac{d}{dx} T_m(x) \right\|_{L^\infty(I)} + \left\| T_m(x) - \tilde{T}_{m,\theta}(x) \right\|_{W^{1,\infty}(I)} \\ &\leq m^2 + \theta \leq m^2 + 1. \end{aligned}$$

where it was used, that  $\frac{d}{dx} T_n(x) \leq n^2$  for all  $x \in I$  (cf. [[26], (1.24)]). Now consider the derivatives of  $T_\ell(x)$  and  $\tilde{T}_{\ell,\delta}(x)$ :

$$\begin{aligned} &\left\| \frac{d}{dx} T_\ell(x) - D \left[ 2 \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) - 1 \right] \right\|_{L^\infty(I)} \\ &= \left\| \frac{d}{dx} (2T_{\ell/2}(x)T_{\ell/2}(x) - 1) - D \left[ 2 \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) - 1 \right] \right\|_{L^\infty(I)} \\ &= \left\| 2 \left( \frac{d}{dx} (T_{\ell/2}(x)T_{\ell/2}(x)) \right) - D \left[ 2 \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) - 1 \right] \right\|_{L^\infty(I)} \\ &\leq \left\| \frac{d}{dx} (T_{\ell/2}(x)T_{\ell/2}(x)) - 2 \left[ D \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right]_1 \frac{d}{dx} \tilde{T}_{\ell/2,\theta}(x) \right\|_{L^\infty(I)} \\ &\quad + \left\| \frac{d}{dx} (T_{\ell/2}(x)T_{\ell/2}(x)) - 2 \left[ D \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right]_2 \frac{d}{dx} \tilde{T}_{\ell/2,\theta}(x) \right\|_{L^\infty(I)} \\ &\leq \left\| 2T_{\ell/2}(x) \left( \frac{d}{dx} T_{\ell/2}(x) \right) - 2 \left[ D \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right]_1 \frac{d}{dx} \tilde{T}_{\ell/2,\theta}(x) \right\|_{L^\infty(I)} \\ &\quad + \left\| 2T_{\ell/2}(x) \left( \frac{d}{dx} T_{\ell/2}(x) \right) - 2 \left[ D \prod_{\theta,2}^2 (\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right]_2 \frac{d}{dx} \tilde{T}_{\ell/2,\theta}(x) \right\|_{L^\infty(I)} \\ &\leq \left\| 2 \underbrace{\frac{d}{dx} T_{\ell/2}(x)}_{\leq (\ell/2)^2} \underbrace{(T_{\ell/2}(x) - \tilde{T}_{\ell/2,\theta}(x))}_{\leq \theta \text{ by IH}} \right\|_{L^\infty(I)} \end{aligned}$$

$$\begin{aligned}
 & + \left\| \underbrace{2\tilde{T}_{\ell/2,\theta}(x)}_{\leq 2} \underbrace{\left( \frac{d}{dx}T_{\ell/2}(x) - \frac{d}{dx}\tilde{T}_{\ell/2,\theta}(x) \right)}_{\leq (\ell/2)^2 - (\ell/2)^2 - \theta} \right\|_{L^\infty(I)} \\
 & + \left\| \underbrace{2 \left( \tilde{T}_{\ell/2,\theta}(x) - \left[ D\tilde{\Pi}_{\theta,2}^2(\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right]_1 \right)}_{\leq \theta \text{ by Prop. 2.6}} \underbrace{\frac{d}{dx}\tilde{T}_{\ell/2,\theta}(x)}_{\leq (\ell/2)^2 + 1} \right\|_{L^\infty(I)} \\
 & + \left\| \underbrace{2 \frac{d}{dx}T_{\ell/2}(x)}_{\leq (\ell/2)^2} \underbrace{\left( T_{\ell/2}(x) - \tilde{T}_{\ell/2,\theta}(x) \right)}_{\leq \theta \text{ by IH}} \right\|_{L^\infty(I)} \\
 & + \left\| \underbrace{2\tilde{T}_{\ell/2,\theta}(x)}_{\leq 2} \underbrace{\left( \frac{d}{dx}T_{\ell/2}(x) - \frac{d}{dx}\tilde{T}_{\ell/2,\theta}(x) \right)}_{\leq (\ell/2)^2 - (\ell/2)^2 - \theta} \right\|_{L^\infty(I)} \\
 & + \left\| \underbrace{2 \left( \tilde{T}_{\ell/2,\theta}(x) - \left[ D\tilde{\Pi}_{\theta,2}^2(\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right]_2 \right)}_{\leq \theta \text{ by Prop. 2.6}} \underbrace{\frac{d}{dx}\tilde{T}_{\ell/2,\theta}(x)}_{\leq (\ell/2)^2 + 1} \right\|_{L^\infty(I)} \\
 & \leq 2(\ell/2)^2\theta + 4\theta + 2(\ell/2)^2\theta + 2\theta + 2(\ell/2)^2\theta + 4\theta + 2(\ell/2)^2\theta + 2\theta \\
 & = 2\ell^2\theta + 12\theta \leq 4\ell^2\theta \leq 4 \cdot 2^{2k+2}\theta = 2^{2k+4}\theta = \delta,
 \end{aligned}$$

where  $D$  is the Jacobian and it was used that  $\ell \leq 2^{k+1}$ ,  $6 \leq \ell^2$  and  $\theta = 2^{-2k-4}\delta$ . The case that  $\ell$  is odd is analogous, using  $\tilde{T}_{\lceil \ell/2 \rceil, \theta}$  in the first three summands and  $\tilde{T}_{\lfloor \ell/2 \rfloor, \theta}$  in the last three summands instead of  $\tilde{T}_{\ell/2, \theta}$ . Now, similarly, consider:

$$\begin{aligned}
 \|T_\ell(x) - \tilde{T}_{\ell,\delta}(x)\|_{L^\infty(I)} & \leq \left\| 2T_{\ell/2}(x) \left( T_{\ell/2}(x) - \tilde{T}_{\ell/2,\theta}(x) \right) \right\|_{L^\infty(I)} \\
 & + \left\| 2\tilde{T}_{\ell/2,\theta}(x) \left( T_{\ell/2}(x) - \tilde{T}_{\ell/2,\theta}(x) \right) \right\|_{L^\infty(I)} \\
 & + \underbrace{\left\| 2\tilde{T}_{\ell/2,\theta}(x)\tilde{T}_{\ell/2,\theta}(x) - 2\tilde{\Pi}_{\theta,2}^2(\tilde{T}_{\ell/2,\theta}(x), \tilde{T}_{\ell/2,\theta}(x)) \right\|_{L^\infty(I)}}_{\leq 2\theta}
 \end{aligned}$$

$$\begin{aligned}
 &\leq \underbrace{\|2T_{\ell/2}(x)\|_{L^\infty(I)}}_{\leq 2} \cdot \underbrace{\|T_{\ell/2}(x) - \tilde{T}_{\ell/2,\theta}(x)\|_{L^\infty(I)}}_{\leq \theta} \\
 &\quad + \underbrace{\|2\tilde{T}_{\ell/2,\theta}(x)\|_{L^\infty(I)}}_{\leq 4} \cdot \underbrace{\|T_{\ell/2}(x) - \tilde{T}_{\ell/2,\theta}(x)\|_{L^\infty(I)}}_{\leq \theta} + 2\theta \\
 &\leq 8\theta \leq \delta.
 \end{aligned}$$

Again, the case for odd  $\ell$  is analogous. With this it follows that

$$\|T_\ell(x) - \tilde{T}_{\ell,\delta}(x)\|_{W^{1,\infty}(I)} \leq \delta.$$

This finishes the construction of  $\Psi_\delta^{k+1}$ . For a sketch of the DNN structure of  $\Psi_\delta^{k+1}$  see Fig. 1. Now consider the construction of  $\Phi_\delta^{\text{Cheb},n}$ :

Define  $\Phi^{3,n}$  as a DNN of depth 0 that emulates the linear map  $\mathbb{R}^{2^k+2k-1} \rightarrow \mathbb{R}^n$  that fulfills

$$\Phi^{3,n}(z_1, \dots, z_{2^k+2k-1})_1 = z_2, \quad \Phi^{3,n}(z_1, \dots, z_{2^k+2k-1})_2 = z_3,$$

and for all  $\ell = 3, \dots, n$ , with  $j := \lceil \log_2(\ell) \rceil$ :

$$\Phi^{3,n}(z_1, \dots, z_{2^k+2k-1})_\ell = z_{\ell+2j-1}.$$

If  $n = 1$  then define  $\Phi_\delta^{\text{Cheb},1} := ((A, b))$ , where  $A \in \mathbb{R}^{1 \times 1}$  and  $b \in \mathbb{R}$  are given by  $A_{1,1} = 1$  and  $b_1 = 0$ . For  $n \geq 2$ , let  $k := \lceil \log_2(n) \rceil$ . Let  $\{\ell_j\}_{j=1}^k \subset \mathbb{N}$  such that  $\text{depth}(\Psi_\delta^k) + 1 = \text{depth}(\Psi_\delta^j) + \ell_j$  for  $j = 1, \dots, k$ , and thus  $\ell_j \leq \max_{j=1}^k \text{depth}(\Psi_\delta^j) = \text{depth}(\Psi_\delta^k)$ . Now define

$$\Phi_\delta^{\text{Cheb},n} := \Phi^{3,n} \circ (\Psi_\delta^1 \circ \Phi_{1,\ell_1}^{\text{Id}}, \dots, \Psi_\delta^k \circ \Phi_{1,\ell_k}^{\text{Id}}),$$

which realizes  $\forall \ell = 1, \dots, n$

$$\left(\Phi_\delta^{\text{Cheb},n}(x)\right)_\ell = \tilde{T}_{\ell,\delta}(x), \quad x \in I, \ell \in \{1, \dots, n\}.$$

See Fig. 2 for a sketch of the construction of  $\Phi_\delta^{\text{Cheb},n}$ . Since neither  $\Phi^{3,n}$  nor any  $\Phi_{1,\ell_i}^{\text{Id}}$  for  $i \in \{1, \dots, k\}$  produce an error, the accuracy of  $\Phi_\delta^{\text{Cheb},n}$  is determined solely by the accuracy of  $\Psi_\delta^1, \dots, \Psi_\delta^k$ . Hence the claim follows.

For the bounds on depth and size recall the impact concatenation and parallelization have on depth and size as described in Definition 1.2. First consider the depth: It holds that

$$\text{depth}(\Psi_\delta^k) \leq C_\Psi (k^3 + k \log_2(1/\delta)),$$

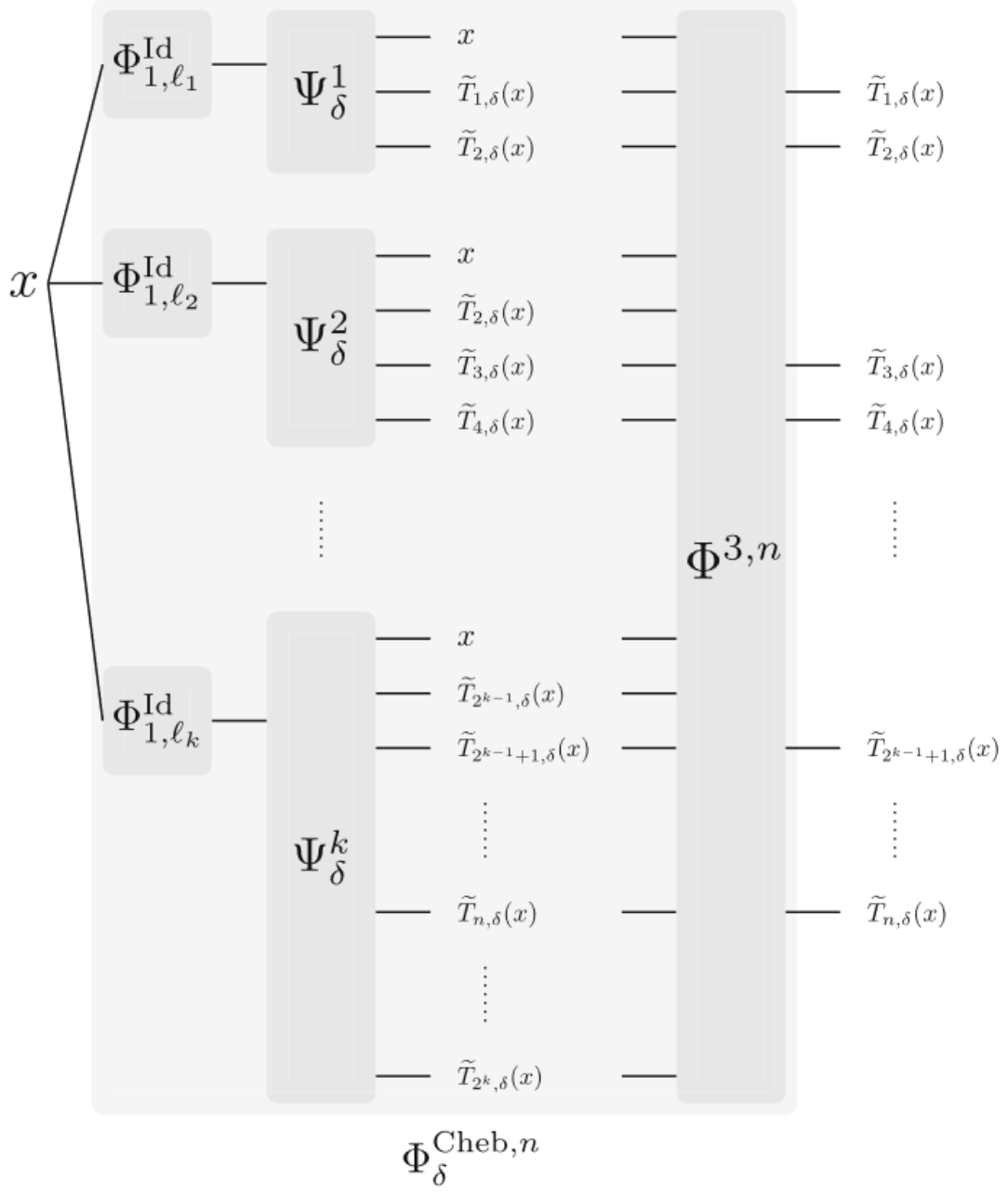


Figure 2: Sketch of  $\Phi_\delta^{\text{Cheb}, n}$  for some  $n \in \mathbb{N}$  and  $\delta \in (0, 1)$ , constructed from identity networks and the previously described  $\Psi_\delta^1, \dots, \Psi_\delta^k$ , with  $k := \lceil \log_2(n) \rceil$ . The subnetwork  $\Phi^{3,n}$  realizes a linear map that selects the desired approximations of univariate Chebyshev polynomials from the outputs of the preceding layer.  
Source: [10]

for some  $C_\Psi$ , which can be shown by induction over  $k$ : For  $k = 1$  it holds that

$$\text{depth}(\Psi_\delta^1) = \text{depth}((\Phi_{1,L_1}^{\text{Id}}, \Phi_{1,L_1}^{\text{Id}}, \Phi)) = \text{depth}\left(\tilde{\prod}_{\delta/4,1}^2\right),$$

for which, by Proposition 2.6, it holds that

$$\begin{aligned} \text{depth}\left(\tilde{\prod}_{\delta/4,1}^2\right) &\leq c \left(1 + \log_2(2) \log_2\left(\frac{2}{\delta/4}\right)\right) = c(1 + \log_2(8/\delta)) \\ &= c(1 + 3 + \log_2(1/\delta)) \leq c'(1 + \log_2(1/\delta)) \\ &\leq C_\Psi(1 + \log_2(1/\delta)) \end{aligned}$$

for some constants  $c$  and  $c'$  and choosing  $C_\Psi > c'$ , so the claim clearly holds. For the induction step, assume that the claim holds for  $k$  and let  $\theta := 2^{-2k-4}$ . Then it holds that

$$\begin{aligned} \text{depth}(\Psi_\delta^{k+1}) &= \text{depth}(\Phi_\delta^{2,k} \circ \Phi^{1,k} \circ \Psi_\theta^k) \\ &= \underbrace{\text{depth}(\Phi_\delta^{2,k})}_{=\text{depth}(\tilde{\prod}_{\theta,2}^2)-1} + \underbrace{\text{depth}(\Phi^{1,k})}_{=0} + \underbrace{\text{depth}(\Psi_\theta^k)}_{\leq C_\Psi(k^3+k\log_2(1/\theta))} - 2 \\ &\leq \tilde{c}(1 + \log_2(2) \log_2(2 \cdot 2^2/\theta)) + C_\Psi(k^3 + k \log_2(1/\theta)) \\ &= \tilde{c}(1 + \log_2(2 \cdot 4 \cdot 2^4 \cdot 2^k/\delta)) + C_\Psi(k^3 + k \log_2(2^{2k+4}/\delta)) \\ &\leq \tilde{c}'(1 + k + \log_2(1/\delta)) + C_\Psi(k^3 + 2k^2 + 4k + k \log_2(1/\delta)) \\ &\leq C_\Psi(k^3 + 3k^2 + 3k + 1 + k \log_2(1/\delta) + \log_2(1/\delta)) \\ &= C_\Psi((k+1)^3 + (k+1) \log_2(1/\delta)), \end{aligned}$$

where the first inequality again follows from Proposition 2.6 for some  $\tilde{c}, \tilde{c}'$  and it was used that  $k^2 \geq 2k$  for all  $k \geq 2$ . For the case  $k = 1$  a stronger base case for the induction was established above, so the claim still holds. Also using w.l.o.g. that  $\tilde{c}' \leq C_\Psi$  is valid, since if this is not the case, one can simply choose the constant to be the maximum of  $C_\Psi$  and  $\tilde{c}'$ .

Finally consider  $\text{depth}(\Phi_\delta^{\text{Cheb},n})$ : For  $n = 1$  the claim follows trivially from definition. For  $n \geq 2$  let  $k := \lceil \log_2(n) \rceil$ .

$$\begin{aligned} \text{depth}(\Phi_\delta^{\text{Cheb},n}) &= \text{depth}(\Phi^{3,n} \circ (\Psi_\delta^1 \circ \Phi_{1,\ell_1}^{\text{Id}}, \dots, \Psi_\delta^k \circ \Phi_{1,\ell_k}^{\text{Id}})) \\ &= \underbrace{\text{depth}(\Phi^{3,n})}_{=0} + \underbrace{\text{depth}((\Psi_\delta^1 \circ \Phi_{1,\ell_1}^{\text{Id}}, \dots, \Psi_\delta^k \circ \Phi_{1,\ell_k}^{\text{Id}}))}_{=\text{depth}(\Psi_\delta^k \circ \Phi_{1,\ell_k}^{\text{Id}})} - 1 \\ &\leq 2 \cdot \text{depth}(\Psi_\delta^k) - 2 \\ &\leq 2 \cdot C_\Psi(k^3 + k \log_2(1/\delta)) \\ &\leq C((1 + \log_2(n))^3 + (1 + \log_2(n)) \log_2(1/\delta)), \end{aligned}$$

which is the claim.

Lastly consider the size: First show, again by induction, that

$$\text{size}(\Psi_\delta^k) \leq C_S (k2^k + 2^k \log_2(1/\delta) + k \log_2(1/\delta) + 2^k + k^3 + k^2 + k)$$

for some  $C_S$ .  
it holds that:

$$\begin{aligned} \text{size}(\Psi_\delta^1) &= \text{size}((\Phi_{1,L_1}^{\text{ID}}, \Phi_{1,L_1}^{\text{ID}}, \Phi)) \\ &= \text{size}(\Phi_{L_1}^{\text{ID}}) + \text{size}(\Phi_{L_1}^{\text{ID}}) + \text{size}(\Phi) \\ &\leq 4 \underbrace{\text{depth} \left( \tilde{\prod}_{\delta/4,1}^2 \right)}_{\leq C_\Psi(1+\log_2(4/\delta))} + 4 + c' (1 + \log_2(1/\delta)) \\ &\leq \underbrace{(12C_\Psi + 4 + c')}_{=:C_1} (1 + \log_2(1/\delta)), \end{aligned}$$

With  $C_1 \leq C_S$ . Now for the bound on  $\text{size}(\Psi_\delta^{k+1})$  first note  $\text{size}(\Phi^{1,k}) = 2 + 2^{k+1}$ ,  $\text{size}(\Phi^{3,n}) = n$  and  $\text{size}(\Phi') = 2 + 2^{k+1}$ . Then it holds that

$$\begin{aligned} \text{size}(\Phi_\delta^{2,k}) &\leq 2 \text{size}(\Phi') + 2 \left( \text{size}(\Phi_{2,L_\theta}^{\text{Id}}) + 2^k \text{size}(\tilde{\prod}_{\theta,2}^2) \right) \\ &\leq 4 + 4 \cdot 2^k + 2 (4L_\theta + 4 + 2^k c (1 + \log_2(2^3/\theta))) \\ &\leq 12 + 4 \cdot 2^k + 8C_\Psi(1 + \log_2(1/\theta)) + 2 \cdot 2^k c (1 + \log_2(2^3/\theta)) \\ &\leq (10 + 3c) \cdot 2^k + (8C_\Psi + 2c2^k)(1 + \log_2(1/\theta)) \\ &\leq \tilde{c} \cdot 2^k + (8C_\Psi + 2c2^k)(5 + 2k + \log_2(1/\delta)), \end{aligned}$$

for some  $\tilde{c}$ . Assuming the claim is true for some  $k$ , it holds that:

$$\begin{aligned} \text{size}(\Psi_\delta^{k+1}) &= \text{size}(\Phi_\delta^{2,k} \circ \Phi^{1,k} \circ \Psi_\theta^k) \\ &\leq 2 \text{size}(\Phi_\delta^{2,k}) + 2 (2 \text{size}(\Phi^{1,k}) + 2 \text{size}(\Psi_\theta^k)) \\ &\leq 2 (\tilde{c}2^k + (8C_\Psi + 2c2^k)(5 + 2k + \log_2(1/\delta))) + 8 + 4 \cdot 2^{k+1} \\ &\quad + 4 (C_S (k2^k + 2^k \log_2(1/\delta) + k \log_2(1/\delta) + 2^k + k^3 + k^2 + k)), \end{aligned}$$

from which - analogously to the depth - it follows, that for some  $C_S$ <sup>8</sup>:

$$\text{size}(\Psi_\delta^{k+1}) \leq C_S ((k+1)2^{(k+1)} + 2^{(k+1)} \log_2(1/\delta) + (k+1) \log_2(1/\delta))$$

---

<sup>8</sup>Again due to the possibility of choosing the maximum of two constants this might not be the same  $C_S$  as above. This vagueness in notation is tolerated to avoid cluttering the notation.

$$+ C_S (2^{(k+1)} + (k+1)^3 + (k+1)^2 + (k+1)).$$

With this the claim for size  $(\Psi_\delta^k)$  follows. Now finally for the claim on size  $(\Phi_\delta^{\text{Cheb},n})$ , it holds that

$$\begin{aligned} \text{size}(\Phi_\delta^{\text{Cheb},n}) &= \text{size}(\Phi^{3,n} \circ (\Psi_\delta^1 \circ \Phi_{1,\ell_1}^{\text{Id}}, \dots, \Psi_\delta^k \circ \Phi_{1,\ell_k}^{\text{Id}})) \\ &\leq 2n + 2 \left( 2 \sum_{i=1}^k \text{size}(\Psi_\delta^i) + 2 \sum_{i=1}^k \text{size}(\Phi_{1,\ell_i}^{\text{Id}}) \right) \\ &\leq 2n + 4 \sum_{i=1}^k \text{size}(\Psi_\delta^i) + 4k \text{depth}(\Psi_\delta^k) + 4k \\ &\leq 2n + 4 \sum_{i=1}^k C_S (i2^i + 2^i \log_2(1/\delta) + i \log_2(1/\delta) + 2^i + i^3 + i^2 + i) \\ &\quad + 4k C_\Psi (k^3 + k \log_2(1/\delta)) + 4k \\ &\leq 2n + 4 \sum_{i=1}^k C_S (i2^i + 2^i \log_2(1/\delta) + i \log_2(1/\delta) + 8 \cdot 2^i) \\ &\quad + 84C_\Psi 2^k + 8C_\Psi 2^k \log_2(1/\delta) + 4k \\ &\leq 2n + 4 \sum_{i=1}^k C_S (i2^i + i \log_2(1/\delta)) + 16C_S n \log_2(1/\delta) \\ &\quad + 64C_S n \log_2(n) + 168C_\Psi n + 16C_\Psi n \log_2(1/\delta) + 4 \log_2(n) + 3 \\ &\leq 2C_S(k^2 + k) (\log_2(1/\delta) + 2^{k+1} - 1) + 64C_S n \log_2(n) \\ &\quad + (168C_\Psi + 2)n + 16n \log_2(1/\delta)(C_\Psi + C_S) + 4 \log_2(n) + 3 \\ &\leq 2C_S(4n + (1 + \log_2(n))) (\log_2(1/\delta) + 2^{k+1} - 1) + 64C_S n \log_2(n) \\ &\quad + (168C_\Psi + 2)n + 16n \log_2(1/\delta)(C_\Psi + C_S) + 4 \log_2(n) + 3 \\ &\leq 10C_S n \log_2(1/\delta) + (10C_S + 4)n - 10C_S n + 64C_S n \log_2(n) \\ &\quad + (168C_\Psi + 2)n + 16n \log_2(1/\delta)(C_\Psi + C_S) + 4 \log_2(n) + 3 \\ &\leq (10C_S + 16(C_\Psi + C_S))n \log_2(1/\delta) \\ &\quad + (9 + 168C_\Psi)n + (64C_S + 4)n \log_2(n) \\ &\leq Cn \log_2(1/\delta) + Cn \log_2(n) + Cn, \end{aligned}$$

for  $C := \max\{10C_S + 16(C_\Psi + C_S), 9 + 168C_\Psi, 64C_S + 4\}$ , which is the claim. Here it was used that  $21 \cdot 2^n \geq n^4$ ,  $4 \cdot 2^n \geq n^3$ ,  $2 \cdot 2^n \geq n^2$ ,  $2^n \geq n$  and  $n \geq 1 + \log_2(n)$ , which is true for all  $n \in \mathbb{N}$ .  $\square$

The previous propositions now allow the postulation of the first main result - the approximation of tensor product Chebyshev polynomials. This will be achieved by



approximating univariate Chebyshev polynomials of degrees  $1, \dots, n$  from Lemma 2.7 and then using the multiplications shown in Proposition 2.6. A sketch of this idea can be seen in Figure 3.

### 2.2.3 Approximation of Tensor Product Chebyshev Polynomials

**Theorem 2.8.** There exists a constant  $C > 0$ , such that for every  $K \in \mathbb{N}$ , every finite subset  $\Lambda \subset \mathbb{N}_0^K$  and every  $\delta \in (0, 1)$  there exists a ReLU DNN  $\Phi_{\Lambda, \delta}$  with input dimension  $K$  and output dimension  $|\Lambda|$ , such that the outputs of  $\Phi_{\Lambda, \delta}$ , which is denoted by  $\{\tilde{T}_{k, \delta}\}_{k \in \Lambda}$ , satisfy

$$\forall k \in \Lambda : \quad \left\| T_k - \tilde{T}_{k, \delta} \right\|_{W^{1, \infty}([-1, 1]^K)} \leq \delta,$$

$$\begin{aligned} \text{depth}(\Phi_{\Lambda, \delta}) &\leq C(1 + \log m_\infty(\Lambda))^3 + C(1 + \log(K) + \log m_\infty(\Lambda)) \log(1/\delta) \\ &\quad + CK \log(m_\infty(\Lambda)) + CK \log K, \\ \text{size}(\Phi_{\Lambda, \delta}) &\leq CK|\Lambda| \log(m_\infty(\Lambda)) + CK|\Lambda| \log(1/\delta) + CK^2|\Lambda| \\ &\quad + CKm_\infty(\Lambda) \log(m_\infty(\Lambda)) + CKm_\infty(\Lambda) \log(1/\delta) + CK^2m_\infty(\Lambda). \end{aligned}$$

*Proof.* To prove existence the DNN  $\Phi_{\Lambda, \delta}$  can explicitly be constructed:  
Let

$$\beta := \frac{1}{2} \delta \underbrace{(1 + \delta)^{-K}}_{\leq 1} \underbrace{K^{-1}}_{\leq 1} \underbrace{(m_\infty(\Lambda)^2 + 1)^{-1}}_{\leq 1} \leq \delta.$$

Now define

$$\Phi_{\Lambda, \delta}^{(2)} := \left( \Phi_{\beta}^{\text{Cheb}, m_\infty(\Lambda)}, \dots, \Phi_{\beta}^{\text{Cheb}, m_\infty(\Lambda)} \right)_d,$$

as the network consisting of the parallelization of  $K$  copies of  $\Phi_{\beta}^{\text{Cheb}, m_\infty(\Lambda)}$ , which are constructed as shown in Lemma 2.7.

For  $\beta' := \frac{1}{2} \delta (m_\infty(\Lambda)^2 + 1)^{-1} \leq \delta$ , define the network  $\Phi_{\Lambda, \delta}^{(1)}$  as the tensor products of univariate Chebyshev polynomials using networks from Proposition 2.6. Denoting the output of  $\Phi_{\Lambda, \delta}^{(2)}$  by  $\{\tilde{T}_{k, \beta}(y_j)\}_{k=1, \dots, m_\infty(\Lambda)}^{j=1, \dots, K}$ , it then follows for an input  $y = (y_1, \dots, y_K) \in [-1, 1]^K$  that

$$\Phi_{\Lambda, \delta} := \Phi_{\Lambda, \delta}^{(1)} \circ \Phi_{\Lambda, \delta}^{(2)}(y_1, \dots, y_K) = \left( \left\{ \prod_{\beta', 1+\delta}^K \left( \tilde{T}_{k_1, \beta}(y_1), \dots, \tilde{T}_{k_K, \beta}(y_K) \right) \right\}_{k \in \Lambda} \right),$$

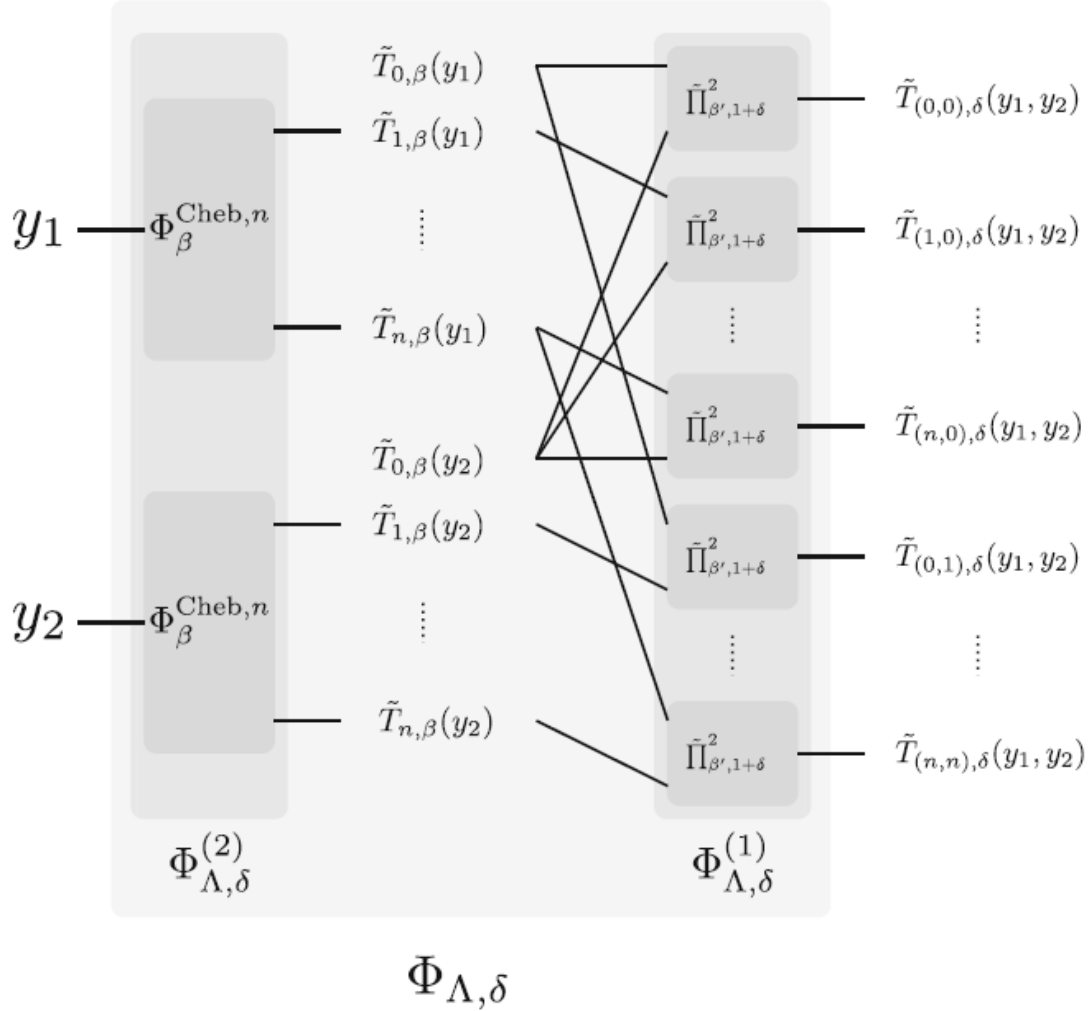


Figure 3: Sketch of  $\Phi_{\Lambda, \delta}$  for  $\Lambda = \{0, \dots, n\}^K$ , with  $n \in \mathbb{N}$ ,  $K = 2$ , and  $\delta \in (0, 1)$ . The subnetwork  $\Phi_\beta^{\text{Cheb}, n}$  from Lemma 2.7 approximates univariate Chebyshev polynomials of degrees  $1, \dots, n$ , which are then multiplied by the networks  $\tilde{\Pi}_{\beta', 1+\delta}^2$  from Proposition 2.6 to obtain approximations of tensor product Chebyshev polynomials. The Chebyshev polynomial of degree 0 is identically equal to 1 and implemented by a bias in the first layer of the product networks  $\tilde{\Pi}_{\beta', 1+\delta}^2$ . Source: [10]

where for  $k_j = 0$ , the factor  $\tilde{T}_{0,\beta} \equiv 1$  is implemented by a bias in the first layer of  $\prod_{\beta', 1+\delta}^K$ . Note that

$$\begin{aligned} \left\| \tilde{T}_{k_j, \beta} \right\|_{L^\infty([-1, 1])} &\leq \left\| T_{k_j} \right\|_{L^\infty([-1, 1])} + \left\| T_{k_j} - \tilde{T}_{k_j, \beta} \right\|_{L^\infty([-1, 1])} \\ &\leq 1 + \beta \leq 1 + \delta \end{aligned}$$

so indeed the assumptions of proposition 2.6 are fulfilled and  $\left\{ \tilde{T}_{k_j, \beta}(y_j) \right\}_{j=1}^K$  can be used as inputs of  $\prod_{\beta', 1+\delta}^K$ .

This concludes the construction of the network  $\Phi_{\Lambda, \delta}$  and it remains to show the claimed error estimates and the bounds for size and depth.

To estimate the error use the triangle inequality and the results of proposition 2.6:

$$\begin{aligned} &\sup_{y \in [-1, 1]^K} \left| T_k(y) - \tilde{T}_{k, \delta}(y) \right| \\ &\leq \sup_{y \in [-1, 1]^K} \left| T_k(y) - \prod_{j=1}^K \tilde{T}_{k_j, \beta}(y_j) \right| \\ &\quad + \sup_{y \in [-1, 1]^K} \left| \prod_{j=1}^K \tilde{T}_{k_j, \beta}(y_j) - \prod_{\beta', 1+\delta}^K \left( \left\{ \tilde{T}_{k_j, \beta}(y_j) \right\}_{j=1}^K \right) \right| \\ &\leq \sup_{y \in [-1, 1]^K} \sum_{i=1}^K \left| \prod_{j=1, \dots, i-1} \tilde{T}_{k_j, \beta}(y_j) \right| \cdot \left| T_{k_i}(y_i) - \tilde{T}_{k_i, \beta}(y_i) \right| \cdot \left| \prod_{j=i+1, \dots, K} T_{k_j}(y_j) \right| \\ &\quad + \beta' \\ &\leq \left( \sum_{i=1}^K (1 + \delta)^{i-1} \beta \right) + \beta' \leq K(1 + \delta)^K \beta + \beta' \leq \delta. \end{aligned}$$

For the estimate on the error in the derivative, consider only the derivative with respect to  $y_1$ . The derivatives with respect to the other inputs satisfy analogous bounds. Using that

$$\begin{aligned} \left| \tilde{T}_{k_j, \beta} \right|_{W^{1, \infty}([-1, 1])} &\leq \left| T_{k_j} \right|_{W^{1, \infty}([-1, 1])} + \left| T_{k_j} - \tilde{T}_{k_j, \beta} \right|_{W^{1, \infty}([-1, 1])} \\ &\leq m_\infty(\Lambda)^2 + \beta \leq m_\infty(\Lambda)^2 + 1, \end{aligned}$$

it follows that

$$\begin{aligned}
 & \text{ess sup}_{y \in [-1,1]^K} \left| \frac{\partial}{\partial y_1} T_k(y) - \frac{\partial}{\partial y_1} \tilde{T}_{k,\delta}(y) \right| \\
 & \leq \text{ess sup}_{y \in [-1,1]^K} \left| \frac{\partial}{\partial y_1} T_k(y) - \frac{\partial}{\partial y_1} \prod_{j=1}^K \tilde{T}_{k_j,\beta}(y_j) \right| \\
 & + \text{ess sup}_{y \in [-1,1]^K} \left| \frac{\partial}{\partial y_1} \prod_{j=1}^K \tilde{T}_{k_j,\beta}(y_j) - \frac{\partial}{\partial y_1} \tilde{\Pi}_{\beta',1+\delta}^K \left( \left\{ \tilde{T}_{k_j,\beta}(y_j) \right\}_{j=1}^K \right) \right| \\
 & \leq \text{ess sup}_{y \in [-1,1]^K} \left| \frac{\partial}{\partial y_1} T_{k_1}(y_1) - \frac{\partial}{\partial y_1} \tilde{T}_{k_1,\beta}(y_1) \right| \cdot \left| \prod_{j=2,\dots,K} T_{k_j}(y_j) \right| \\
 & + \text{ess sup}_{y \in [-1,1]^K} \sum_{i=2,\dots,K} \left| \frac{\partial}{\partial y_1} \tilde{T}_{k_1,\beta}(y_1) \right| \cdot \left| \prod_{j=2,\dots,i-1} \tilde{T}_{k_j,\beta}(y_j) \right| \\
 & \cdot \left| T_{k_i}(y_i) - \tilde{T}_{k_i,\beta}(y_i) \right| \cdot \left| \prod_{j=i+1,\dots,K} T_{k_j}(y_j) \right| \\
 & + \text{ess sup}_{y \in [-1,1]^K} \prod_{j=2,\dots,K} \tilde{T}_{k_j,\beta}(y_j) - \left( \frac{\partial}{\partial x_1} \tilde{\Pi}_{\beta',1+\delta}^K \right) \left( \left\{ \tilde{T}_{k_j,\beta}(y_j) \right\}_{j=1}^K \right) \cdot \\
 & \cdot \left| \frac{\partial}{\partial y_1} \tilde{T}_{k_1,\beta}(y_1) \right| \\
 & \leq \beta + \left( \sum_{i=2}^K (m_\infty(\Lambda)^2 + 1) (1 + \delta)^{i-2} \beta \right) + \beta' (m_\infty(\Lambda)^2 + 1) \\
 & \leq K(1 + \delta)^K \beta (m_\infty(\Lambda)^2 + 1) + \beta' (m_\infty(\Lambda)^2 + 1) \\
 & \leq \frac{\delta}{2} + \frac{\delta}{2} \leq \delta,
 \end{aligned}$$

which, using the definition of the Sobolev norm defined in the notation 1.5, finishes the proof of the error estimate.

The bounds for size and depth are - by Lemma 2.7 and proposition 2.6 - then given by:

$$\begin{aligned}
\text{depth}(\Phi_{\Lambda,\delta}) &\leq \text{depth}\left(\Phi_{\Lambda,\delta}^{(1)}\right) + 1 + \text{depth}\left(\Phi_{\Lambda,\delta}^{(2)}\right) \\
&\leq \text{depth}\left(\prod_{\beta', 1+\delta}^K\right) + 1 + \text{depth}\left(\Phi_{\beta}^{\text{Cheb}, m_{\infty}(\Lambda)}\right) \\
&\leq C\left(1 + \log(K) \log\left(K(1+\delta)^K/\beta'\right)\right) + 1 \\
&\quad + C\left(1 + \log m_{\infty}(\Lambda)\right) \log(1/\beta) + C\left(1 + \log m_{\infty}(\Lambda)\right)^3 \\
&\leq C\left(\log(K)^2 + \log(K)K \log(1+\delta) + \log(K) \log(m_{\infty}(\Lambda))\right) \\
&\quad + C\left(\log(K) \log(1/\delta) + (1 + \log m_{\infty}(\Lambda))^2\right) \\
&\quad + (\log(K) + K \log(1+\delta))(1 + \log m_{\infty}(\Lambda)) \\
&\quad + (1 + \log m_{\infty}(\Lambda)) \log(1/\delta) + C\left(1 + \log m_{\infty}(\Lambda)\right)^3 \\
&\leq C\left(1 + \log m_{\infty}(\Lambda)\right)^3 + C\left(1 + \log(K) + \log m_{\infty}(\Lambda)\right) \log(1/\delta) \\
&\quad + CK \log(m_{\infty}(\Lambda)) + CK \log K,
\end{aligned}$$

and

$$\begin{aligned}
\text{size}(\Phi_{\Lambda,\delta}) &\leq C\left[\text{size}\left(\Phi_{\Lambda,\delta}^{(1)}\right) + \text{size}\left(\Phi_{\Lambda,\delta}^{(2)}\right)\right] \\
&\leq C\left[|\Lambda| \text{size}\left(\prod_{\beta', 1+\delta}^K\right) + K \text{size}\left(\Phi_{\beta}^{\text{Cheb}, m_{\infty}(\Lambda)}\right)\right] \\
&\leq C\left[|\Lambda| \left(1 + K \log\left(K(1+\delta)^K/\beta'\right)\right) + K(m_{\infty}(\Lambda) \log(1/\beta) \right. \\
&\quad \left. + m_{\infty}(\Lambda)(1 + \log m_{\infty}(\Lambda)))\right] \\
&\leq C\left[|\Lambda| \left(K \log(K) + K^2 \log(1+\delta) + K \log(m_{\infty}(\Lambda)) + K \log(1/\delta)\right) \right. \\
&\quad \left. + K(m_{\infty}(\Lambda) \log(m_{\infty}(\Lambda)) + m_{\infty}(\Lambda) \log(1/\delta) \right. \\
&\quad \left. + (\log(K) + K \log(1+\delta))m_{\infty}(\Lambda) + m_{\infty}(\Lambda)(1 + \log m_{\infty}(\Lambda)))\right] \\
&\leq C\left[K|\Lambda| \log(m_{\infty}(\Lambda)) + K|\Lambda| \log(1/\delta) + K^2|\Lambda| \right. \\
&\quad \left. + Km_{\infty}(\Lambda) \log(m_{\infty}(\Lambda)) + Km_{\infty}(\Lambda) \log(1/\delta) + K^2m_{\infty}(\Lambda)\right],
\end{aligned}$$

as claimed.  $\square$

### 2.3 Construction of ReLU DNN Approximations of Multivariate Holomorphic Functions

For the second main result the preparations in Chapter 1, namely Lemma 2.3, are combined with the first main result, Theorem 2.8. The outcome is a the approximation of multivariate holomorphic functions (with a possibly corrupted

approximation) through a DNN with a bound on the error in both the  $L^\infty$  norm and the  $W^{1,\infty}$ -Sobolev norm as well as bounds on the depth and size of the DNN.

**Theorem 2.9.** Let the assumptions of Lemma 2.3 be true. Then there exists a ReLU DNN  $\Phi_n^{\check{f}}$  for all  $n \in \mathbb{N}$  with  $K$ -dimensional input and one-dimensional output, such that for all  $\rho' \in (1, \rho)$ , there exists a constant  $C(K, \rho, \rho') > 0$  and a constant  $c(\rho) > 0$ , such that:

$$\begin{aligned} \|f - \Phi_n^{\check{f}}\|_{L^\infty([-1,1]^K)} &\leq C \left( \max_{z \in \mathcal{E}_\varrho} |f(z)| + \|f - \check{f}\|_{L^\infty([-1,1]^K)} \right) \rho'^{-n} \\ &\quad + \left( \frac{2}{\pi} \log(n+1) + 1 \right)^K \|f - \check{f}\|_{L^\infty([-1,1]^K)}, \\ \|f - \Phi_n^{\check{f}}\|_{W^{1,\infty}([-1,1]^K)} &\leq C \left( \max_{z \in \mathcal{E}_\varrho} |f(z)| + \|f - \check{f}\|_{L^\infty([-1,1]^K)} \right) \rho'^{-n} \\ &\quad + n^2 \left( \frac{2}{\pi} \log(n+1) + 1 \right)^K \|f - \check{f}\|_{L^\infty([-1,1]^K)}, \\ \text{depth}(\Phi_n^{\check{f}}) &\leq cKn(1 + \log(Kn)), \quad \text{size}(\Phi_n^{\check{f}}) \leq cK^2(n+1)^{K+1}. \end{aligned}$$

The weights and biases depend solely on  $f$  by  $\{\check{f}(x_j^n)\}_{j \in \{0, \dots, n\}^K}$ , where  $x_j^n = \cos(j\pi/n)$ . These are  $(n+1)^K$  nodes.

*Proof.* Use the approximation  $\check{p}_{f,n}$  of  $f$  as in (5). Use 2.8 with  $\Lambda = \{0, \dots, n\}^K$ ,  $|\Lambda| = (n+1)^K$ ,  $m_\infty(\Lambda) = n$  and  $\delta = \rho^{-n}(n+1)^{-K}$  to approximate the tensor product Chebyshev polynomials.

Denote by  $k^{(\ell)} \in \Lambda$  the element satisfying  $(\Phi_{\Lambda, \delta})_\ell = \tilde{T}_{k^{(\ell)}, \delta}$ , where  $\tilde{T}_{k^{(\ell)}, \delta}$  are the outputs of the network given by Proposition 2.8. Now define  $A \in \mathbb{R}^{1 \times |\Lambda|}$ , such that  $A_{1, \ell} = \check{f}_{k^{(\ell)}; n}$  for  $\ell = 1, \dots, |\Lambda|$ . Let  $b := 0 \in \mathbb{R}$  and define

$$\Phi_n^{\check{f}} := ((A, b)) \circ \Phi_{\Lambda, \delta}.$$

Note:  $\text{depth}(((A, b))) = 0$  and  $\text{size}(((A, b))) \leq |\Lambda|$ . For some  $C(K, \rho) > 0$  the realization of  $\Phi_n^{\check{f}}$  satisfies:

$$\begin{aligned} \Phi_n^{\check{f}} &= \sum_{k \in \Lambda} \check{f}_{k; n} \tilde{T}_{k, \delta}, \\ \|\check{p}_{f,n} - \Phi_n^{\check{f}}\|_{W^{1,\infty}([-1,1]^K)} &\leq \sum_{k \in \Lambda} |\check{f}_{k; n}| \|T_k - \tilde{T}_{k, \delta}\|_{W^{1,\infty}([-1,1]^K)} \\ &\leq \delta \sum_{k \in \Lambda} |\check{f}_{k; n}|, \end{aligned}$$

which, using Corollary 2.4, can be estimated as

$$\begin{aligned} &\leq \delta C \max_{z \in \mathcal{E}_\varrho} |f(z)| + \delta(n+1)^K \pi^K \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\ &\leq C \rho^{-n} \left( \max_{z \in \mathcal{E}_\varrho} |f(z)| + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \right). \end{aligned}$$

For some  $C(K, \rho) > 0$  an error bound can be found by using Lemma 2.3:

$$\begin{aligned} &\|f - \Phi_n^{\check{f}}\|_{L^\infty([-1,1]^K)} \\ &\leq \|f - \check{p}_{f,n}\|_{L^\infty([-1,1]^K)} + \|\check{p}_{f,n} - \Phi_n^{\check{f}}\|_{L^\infty([-1,1]^K)} \\ &\leq \left(1 + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty}\right) K \left(\frac{2\rho}{\rho-1}\right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \rho^{-n-1} \\ &\quad + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\ &\quad + C \rho^{-n} \left( \max_{z \in \mathcal{E}_\varrho} |f(z)| + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \right) \\ &\leq C \left( \max_{z \in \mathcal{E}_\varrho} |f(z)| + \|f - \check{f}\|_{L^\infty([-1,1]^K)} \right) \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \rho^{-n} \\ &\quad + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\ &\quad C \left( \max_{z \in \mathcal{E}_\varrho} |f(z)| + \|f - \check{f}\|_{L^\infty([-1,1]^K)} \right) \rho'^{-n} \\ &\quad + \left( \frac{2}{\pi} \log(n+1) + 1 \right)^K \|f - \check{f}\|_{L^\infty([-1,1]^K)}, \end{aligned}$$

as claimed. The last inequality stems from the bound on  $\|I_n^{\text{CC}}\|_{L^\infty, L^\infty}$  as stated in Chapter 2.1.2.

Analogously, using Lemma 2.3, for every  $\rho' \in (1, \rho)$  and with  $C'(s = 1, \rho, \rho') > 0$  it follows with the estimation on the realization from the beginning of this proof,

that there exists  $C(K, \rho, \rho') > 0$ , such that:

$$\begin{aligned}
 & \|f - \Phi_n^{\check{f}}\|_{W^{1,\infty}([-1,1]^K)} \\
 & \leq \|f - \check{p}_{f,n}\|_{W^{1,\infty}([-1,1]^K)} + \|\check{p}_{f,n} - \Phi_n^{\check{f}}\|_{W^{1,\infty}([-1,1]^K)} \\
 & \leq \left(1 + \|I_n^{\text{CC}}\|_{W^{1,\infty}, W^{1,\infty}}\right) K \left(\frac{2C'\rho'}{\rho' - 1}\right)^K \max_{z \in \mathcal{E}_\varrho} |f(z)| \rho'^{-n-1} \\
 & \quad + \|I_n^{\text{CC}}\|_{L^\infty, W^{1,\infty}} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\
 & \quad + C\rho^{-n} \left(\max_{z \in \mathcal{E}_\varrho} |f(z)| + \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)}\right) \\
 & \leq C\rho'^{-n} \|I_n^{\text{CC}}\|_{W^{1,\infty}, W^{1,\infty}} \max_{z \in \mathcal{E}_\varrho} |f(z)| + C\rho^{-n} \|I_n^{\text{CC}}\|_{L^\infty, L^\infty} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\
 & \quad + \|I_n^{\text{CC}}\|_{L^\infty, W^{1,\infty}} \|f - \check{f}\|_{L^\infty([-1,1]^K)} \\
 & \leq C \left(\max_{z \in \mathcal{E}_\varrho} |f(z)| + \|f - \check{f}\|_{L^\infty([-1,1]^K)}\right) \rho'^{-n} \\
 & \quad + n^2 \left(\frac{2}{\pi} \log(n+1) + 1\right)^K \|f - \check{f}\|_{L^\infty([-1,1]^K)},
 \end{aligned}$$

as claimed. The last inequality stems from the bounds on  $\|I_n^{\text{CC}}\|_{L^\infty, L^\infty}$  and on  $\|I_n^{\text{CC}}\|_{W^{1,\infty}, W^{1,\infty}}$  as stated in Chapter 2.1.2.

With Definition 1.2 and Theorem 2.8 it follows that there exists  $c(\rho) > 0$ , such that for every  $n \in \mathbb{N}$ :

$$\begin{aligned}
 \text{depth}(\Phi_n^{\check{f}}) & \leq \text{depth}(((A, b))) + 1 + \text{depth}(\Phi_{\Lambda, \delta}) \\
 & \leq 1 + c(1 + \log m_\infty(\Lambda))^3 + c(1 + \log(K) + \log m_\infty(\Lambda)) \log(1/\delta) \\
 & \quad + cK \log(m_\infty(\Lambda)) + cK \log(K) \\
 & \leq cKn(1 + \log(Kn)),
 \end{aligned}$$

and similarly:

$$\begin{aligned}
 \text{size}(\Phi_n^{\check{f}}) & \leq c \text{size}(((A, b))) + c \text{size}(\Phi_{\Lambda, \delta}) \\
 & \leq c|\Lambda| + c|\Lambda|K \log(m_\infty(\Lambda)) + c|\Lambda|K \log(1/\delta) + c|\Lambda|K^2 \\
 & \quad + cKm_\infty(\Lambda) \log(m_\infty(\Lambda)) + cKm_\infty(\Lambda) \log(1/\delta) + cK^2m_\infty(\Lambda) \\
 & \leq cK^2(n+1)^{K+1},
 \end{aligned}$$

as claimed. □



### 3 Implementation

All code is published on GitHub<sup>9</sup>. The implementation was done in Python 3.10.8. Auxiliary files can be found in the *utils* folder, while the testing and result files are in the main folder of the repository.

#### 3.1 Implementing the Chebyshev Coefficients

To compute the Chebyshev coefficients of a given function  $f$  on  $[-1, 1]^d$  the class *Chebyshev* was written, that can be found in *utils/chebyshev.py*. The class has the following attributes:

- `fun`: The function  $f$  that is to be approximated.
- `degree`: The degree of the Chebyshev approximation.
- `dim`: The dimension of the input of  $f$ .
- `grid`: The grid of the Chebyshev approximation.

This class enables the computation of the coefficients  $\check{f}_{k;n}$  as described in Chapter 2.1.3, that is done by the function *coefficients* shown in Code Block 1. Note that the coefficients are calculated assuming the knowledge of the function to be approximated. Since this will often not be the case, but instead only a number of gridpoints and the values at those gridpoints will be known, it is also possible to interpolate those point-value-pairs with some approximation method like linear spline approximation and use this approximated function as input *fun* for an instance of the class.

```

1 def coefficients(self, reshape = False):
2     # reshape: if True, reshape the coefficients to the shape of
3     # the grid
4     # returns grid and coefficients of the interpolation as in
5     # Chapter 2
6     fac = self._indicator()
7     fac = np.array([2**i for i in fac])
8     fac = fac*(2*self.degree)**(-self.dim) #This is 2^{\mathbb{m}{1}}
9     _{S(n)}(k)} * (2n)^{-k}
10    grid_j = self._grid(2*self.degree-1)
11    grid_cos_j = self._cos_grid(grid_j)
12    grid_cos_kj = self._cos_kj(self.grid, grid_j)
13    f_hat = self.fun(grid_cos_j) # Transpose for correct input
14    if self.dim == 1:

```

<sup>9</sup>[https://github.com/FaBremer/constructed\\_DNNs](https://github.com/FaBremer/constructed_DNNs)

```

12     f_hat = f_hat.reshape((2*self.degree,))
13     f_hat = self._sum_mult(f_hat, grid_cos_kj)
14     ret = fac*f_hat # elemt-wise multiplication
15     if reshape:
16         self.get_degrees(), ret.reshape([self.degree +1 for i in
range(self.dim)])
17     return self.grid, ret

```

Code Block 1: Computing the Chebyshev coefficients in the Chebyshev class according to Chapter 2.1.3 as can be found in the file *utils/chebyshev.py*

Using the coefficients, the Chebyshev approximation for the multivariate case  $\check{p}_{f,n}$  as described in Chapter 2.1.3 is computed by the function *interpolate* shown in Code Block 2.

```

1 def _interpolate(self, coeff, x):
2     # Interpolates the function at the given point
3     # coeff: coefficients of the interpolation
4     # x: point to evaluate at (single point)
5     # returns the interpolation at x
6     result = 0
7     l, k = self.grid.shape # k is dimension, l-1 is degree of
approximation
8     if k != len(x):
9         raise Exception(f"Lenght of input must match Dimenssion {
self.dim}!")
10    for i in range(l):
11        sub_result = coeff[i]
12        grid_point = self.grid[i]
13        for j, arg in enumerate(x):
14            sub_result *= self.polynomials(grid_point[j], arg)
15        result += sub_result
16    return result
17
18 def interpolate(self, x, coeff = None):
19     # Interpolates the function at multiple given points (with
shape (len(x),dim))
20     # x: points to evaluate at
21     # coeff: coefficients of the interpolation, if None, use self.
coefficients()
22     # returns the interpolations at x
23     if coeff is None:
24         grid, coeff = self.coefficients()
25     k = self.dim
26     if len(x.shape) == 1:
27         x = x.reshape(x.shape[0], k)
28     if k != x.shape[1]:
29         raise Exception(f"Shape of input must be (x,{self.dim})
for x > 1!")

```

```
30 return np.asarray([self._interpolate(coeff, y) for y in x])
```

Code Block 2: Interpolation in the Chebyshev class according to Chapter 2.1.3 as can be found in the file *utils/chebyshev.py*

The coefficients are ready to be stored in a DNN of the form  $(A, b)$  as described in the proof of Theorem 2.9.

## 3.2 Constructing the DNNs

To construct the DNNs a class *NeuralNetwork* was written. The instances of this class are the DNNs. The class has the following attributes:

- `input_dim`: The dimension of the input.
- `output_dim`: The dimension of the output.
- `hidden_dims`: A list of numbers, where each number represents the number of neurons in a hidden layer. This means that for a depth  $L$ , an input dimension  $d$ , an output dimension  $\tilde{d}$  and  $d_i$  for the  $i$ -th item in the list `hidden_dims` of length  $L$ , the first weight matrix  $A_1$  is an element of  $\mathbb{R}^{d_1 \times d}$ , the last weight matrix  $A_{L+1}$  is an element of  $\mathbb{R}^{\tilde{d} \times d_L}$  and the weight matrices  $A_i$  for  $i = 2, \dots, L$  are elements of  $\mathbb{R}^{d_i \times d_{i-1}}$ . The bias vectors  $b_i$  for  $i = 1, \dots, L$  are elements of  $\mathbb{R}^{d_i}$  and the last bias vector  $b_{L+1}$  is an element of  $\mathbb{R}^{\tilde{d}}$ .
- `weights`: A list of the weight matrices  $A_i$  for  $i = 1, \dots, L + 1$ . The weights are initialized with all entries being 0.
- `biases`: A list of the bias vectors  $b_i$  for  $i = 1, \dots, L + 1$ . The biases are initialized with all entries being 0.

The class has the following main methods:

- `realize`: Returns the realization of the DNN for a given input  $x$  using the ReLU activation function.
- `count_depth_size`: Returns the depth and size of the DNN.
- `concatenate`: Returns the concatenation of two DNNs as defined in Definition 1.2.
- `n_parallelize`: Returns the distinct parallelization of  $n$  DNNs as defined in Definition 1.2.
- `n_undistinct_parallelize`: Returns the (undistinct) parallelization of  $n$  DNNs as defined in Definition 1.2.

While the complete construction can be studied in the files in the repository, the following code snippets highlight the most important parts of the construction as well as general methods that were used in the construction of the DNNs. With this class and the methods, the DNNs can be constructed as described in Section 2:

### 3.2.1 Constructing the DNNs for Multiplication

The construction of the DNNs  $\tilde{\chi}_{\delta,M}$  and  $\tilde{\prod}_{\delta,M}^n$  as described in section 2.2.1 can be found in the file *utils/multiplication.py*. The DNNs are constructed as described in the proofs of Lemma 2.5 and Proposition 2.6. To guarantee an exact implementation of the networks that were described in theory, only NumPy [8] and the class *NeuralNetwork* were used and all DNNs implemented as instances of that class. Code Block 3 shows the implementation of the function  $g$  as a DNN, that emulates the sawtooth function as described in the proof of Lemma 2.5 as well as the  $m$ -fold concatenation of  $g$ .

```
1 def g_dnn():
2     # Sawtooth function as DNN
3     # returns DNN that emulates sawtooth function
4     g = nn.NeuralNetwork(1, [3], 1)
5     g.weights[0] = np.array([[2],[4],[2]])
6     g.biases[0] = np.array([0, -2, -2])
7     g.weights[1] = np.array([[1,-1,1]])
8     return g
9
10 def g_m_dnn(m):
11     # m : number of concatenations of g
12     # returns m concatenations of g as DNN
13     g_m = g_dnn()
14     for i in (range(m-1)):
15         A = g_dnn()
16         g_m = g_m.concatenate(A)
17     return g_m
```

Code Block 3: Emulation of the sawtooth function  $g$  as a DNN as can be found in the file *utils/multiplication.py*

As the very basic DNN  $g$  had to be implemented manually as a DNN with input dimension 1, output dimension 1 and one hidden layer with 3 neurons. The weight matrix  $A_1 \in \mathbb{R}^{3 \times 1}$  and the bias vector  $b_1 \in \mathbb{R}^3$  were set as  $A_1 = (2, 4, 2)^\top$  and  $b_1 = (0, -2, -2)^\top$ . The weight matrix  $A_2 \in \mathbb{R}^{1 \times 3}$  was set as  $A_2 = (1, -1, 1)$  and the bias vector  $b_2 \in \mathbb{R}$  was set as  $b_2 = 0$ . The resulting DNN then realizes for an input

$x \in [0, 1]$  and using ReLU as the activation function:

$$g(x) = \max\{0, 2x\} - \max\{0, 4x - 2\} + \max\{0, 2x - 2\} = \begin{cases} 2x & \text{if } x < \frac{1}{2} \\ 2(1 - x) & \text{if } x \geq \frac{1}{2} \end{cases},$$

as desired. Having constructed  $g$  and  $g_m$  allows for the construction of the function  $f_m$  as described in the proof of Lemma 2.5 and consequently to the construction of the DNN  $\tilde{\chi}_{\delta, M}$ , which is implemented by the function *two\_mult\_dnn*.

With this preparation the construction of  $\prod_{\delta, M}^n$  is now implemented by the function *mult\_dnn*, that differentiates the cases  $M = 1$  and  $M > 1$  as described in the proof of Proposition 2.6 and is based on the function  $R^\ell$ , implemented by the function *r\_dnn*. Since the multiplication of  $n$  elements is achieved by repeated pairwise multiplication, the input dimension of  $R^\ell$  should be a power of two. The theory (i.e. Proposition 2.6) solves this by virtually expanding the input vector to a length that is a power of two and filling it with ones, which is to be implemented by the bias in the first layer. As can be seen in Code Block 4 this bears some complications in the actual implementation, since cases have to be differentiated where  $n$  is even and where it is odd and the realization of the multiplication has to be emulated manually for the different cases.

```

1 def r_dnn(l, n_tilde, n, delta):
2     # l: number of concatenations
3     # n_tilde: length of input (needed, aka a power of 2)
4     # n: length of input (actual)
5     # delta: error bound
6     # returns DNN that multiplies n numbers with error \leq delta
7     if n_tilde == n:
8         if l == 1:
9             temp = []
10            if n_tilde == 2:
11                return two_mult_dnn(delta*(n_tilde**(-2)), 2)
12            base = two_mult_dnn(delta*(n_tilde**(-2)), 2)
13            for _ in range(int(n_tilde/2)-1):
14                temp.append(two_mult_dnn(delta*(n_tilde**(-2)), 2)
15            )
16            res = base.n_parallelize(temp)
17            return res
18        else:
19            return r_dnn(l-1, int(.5*n_tilde), n, delta).
20            concatenate(r_dnn(1, n_tilde, n, delta))
21            # If n is not a power of 2, we need to add ones through the
22            # biases to make it work
23        else:
24            res = r_dnn(l, n_tilde, n_tilde, delta)

```

---

```

22         for i in range(n_tilde - n):
23             res.weights[0] = res.weights[0][:, :n]
24             if n%2 != 0:
25                 res.biases[0][3*n-1:3*(n+1)] = [1,-1,1,-1]
26                 for i in range(n, n_tilde-2,2):
27                     res.biases[0][3*(i+1):3*(i+3)] =
[1,-1,1,-1,2,-2]
28             else:
29                 for i in range(n, n_tilde,2):
30                     res.biases[0][3*i:3*(i+2)] = [1,-1,1,-1,2,-2]
31         return res
32
33 def mult_dnn(n, delta=.001, M=1):
34     # n: number of numbers to multiply
35     # delta: error bound
36     # M: numbers to be multiplied \in [-M,M]
37     # returns DNN that multiplies n numbers with error \leq delta
38     if n == 1:
39         return identity_dnn(1)
40     if M == 1:
41         k = 1
42         state = True
43         while state:
44             if 2**k >= n:
45                 n_tilde = 2**k
46                 state = False
47             else:
48                 k += 1
49         return r_dnn(int(np.log2(n_tilde)), n_tilde, n, delta)
50     else:
51         result = mult_dnn(n, delta*(M**(-n)), 1)
52         result.weights[0] = result.weights[0]*(M**(-1))
53         result.weights[-1] = result.weights[-1]*(M**n)
54         result.biases[-1] = result.biases[-1]*(M**n)
55         return result

```

Code Block 4: Emulation of the function  $R^\ell$  as a DNN and implementation of  $\tilde{\prod}_{\delta,M}^n$  as can be found in the file *utils/multiplication.py*

### 3.2.2 Constructing the DNNs for univariate and multivariate Chebyshev Polynomials

With the ability to construct multiplications as DNNs the implementation of the approximation of univariate and multivariate Chebyshev polynomials can now be realized, which can be found in the file *cheb\_dnn.py*. First the construction of  $\Phi_\delta^{\text{Cheb},n}$  as described in Lemma 2.7 is shown in Code Block 5.

---

```

1 def cheb_delta_n(delta, n):
2     # returns \Phi_{\delta}^{Cheb,n} from the proof of Lemma 2.7 as
    DNN
3     if n == 1:
4         result = nn(1, [], 1)
5         result.weights[0][0][0] = 1
6         return result
7     k = int(np.ceil(np.log2(n)))
8     psi_delta_k_ = psi_delta_k(k, delta)
9     depth_psi, _ = psi_delta_k_.count_depth_size()
10    depth_psi_delta_k_plus_one = depth_psi + 1
11    big_parallelization = []
12    for j in range(k):
13        psi_j = psi_delta_k(j+1, delta)
14        depth_psi_j, _ = psi_j.count_depth_size()
15        l_j = depth_psi_delta_k_plus_one - depth_psi_j
16        element = psi_j.concatenate(mult.identity_dnn_n(1, l_j))
17        big_parallelization.append(element)
18    second_part = big_parallelization[0].n_undistinct_parallelize(
        big_parallelization[1:])
19    return phi_3_n(k, n).concatenate(second_part)

```

---

Code Block 5: Implementation of the DNN  $\Phi_{\delta}^{Cheb,n}$  as can be found in the file *cheb\_dnn.py*

The functions  $\Psi_{\delta}^k$ ,  $\Phi^{3,n}$ ,  $\Phi^{1,k}$ ,  $\Phi_{\delta}^{2,k}$  and  $\Phi$  from the proof of Lemma 2.7 are implemented as *psi\_delta\_k*, *phi\_3\_n*, *phi\_1\_k*, *phi\_delta\_2\_k* and *phi* respectively.

The construction of  $\Phi_{\Lambda,\delta}$  as described in Theorem 2.8 is shown in Code Block 6. Since the output of  $\Phi_{\Lambda,\delta}^{(2)}$  is in an order unpractical for the implementation, the function *reorder\_output* is used to reorder the output without an impact on depth and size (depth not affected due to double concatenation and size not affected since the reordering has to happen anyway). The function *m\_infty* is used to calculate  $m_{\infty}$  as described in the proof of Theorem 2.8.

---

```

1 def reorder_output(k, var_lambda):
2     # returns the reordered output of \Phi_{\Lambda,\delta}^{(2)}
    from the proof of Theorem 2.8 as DNN
3     m = m_infty(var_lambda)
4     reorder = nn(k*m, [], len(var_lambda)*k)
5     counter = 0
6     for vector in var_lambda:
7         for j in range(len(vector)):
8             if vector[j] == 0:
9                 counter += 1
10            else:

```

---

```

11         reorder.weights[0][counter][vector[j]-1+j*m] = 1
12         counter += 1
13     # add bias for Chebyshev polynomials of degree 0
14     for i in range(len(reorder.weights[0])):
15         if sum(reorder.weights[0][i]) == 0:
16             reorder.biases[0][i] = 1
17     return reorder
18
19 def phi_Lambda_delta(k, var_lambda, delta =.001):
20     # returns \Phi_{\Lambda,\delta} from the proof of Theorem 2.8
    as DNN
21     phi_2 = phi_Lambda_delta_2(k, var_lambda, delta)
22     ordered_phi_2 = reorder_output(k, var_lambda).concatenate(
    phi_2)
23     return phi_Lambda_delta_1(k, var_lambda, delta).concatenate(
    ordered_phi_2)

```

Code Block 6: Implementation of the DNN  $\Phi_{\Lambda,\delta}$  as can be found in the file *cheb\_dnn.py*

### 3.2.3 Algorithm for the Construction of a DNN for approximating a given Function

As described in the theory in Chapters 1 and 2, the goal of constructing the DNNs is to emulate a map  $f : [-1, 1]^K \rightarrow \mathbb{R}$ , that admits a holomorphic extension to a closed Bernstein polyellipse  $\mathcal{E}_\rho$ , with  $\rho = (\rho, \dots, \rho) \in \mathbb{R}^K$  for some  $\rho > 1$  that will be unknown in general. To implement the algorithm based on the DNN construction in the previous chapters, an approximation  $\tilde{f}$  of  $f$  is needed and will be given by classic Chebyshev approximation as described in Chapter 3.1. With this an algorithm is implemented that will output an approximating DNN whose estimated error is at most some given accuracy  $\varepsilon$  and whose depth and size are bounded by the bounds from the theory in Chapter 2. As a distinguishing feature in contrast to more classical DNN approaches, this algorithm is purely deterministic, and does not depend on loss function minimization by "black-box" algorithms.

For  $n \in \mathbb{N}$ ,  $\Lambda = \{0, \dots, n\}^K$  and  $\delta = (n+1)^{-K} \rho^{-n}$ , the DNN  $\Phi_n^{\tilde{f}} := ((A, b)) \circ \Phi_{\Lambda,\delta}$ , where  $A$  contains the Chebyshev coefficients and  $b = 0$ , has been described in Theorem 2.9. In the proof of this theorem  $\delta = (n+1)^{-K} \rho^{-n}$  is shown to be sufficient. But since a priori  $\rho$  may not be known, the algorithm also provides a way to determine a sufficient value for  $\delta$ .

In conclusion the following algorithm shown in Code Block 7, that can be found in the file *dnn\_construction.py*, computes  $n_* \in \mathbb{N}$  and  $\delta_* \in (0, 1)$  and constructs the ReLU DNN  $\Phi_{n_*, \delta_*}^{\tilde{f}}$  approximating  $f$ . Analogously to the purely theoretical



implementation in [[10], Section 4], the following Proposition based on the actual implementation follows:

**Proposition 3.1.** Let  $C > 0$  not be dependent on  $n_*, \delta_*, K$ . Assuming the complexity of constructing a DNN from its weights and biases and evaluating a DNN are both proportional to the number of nonzero network weights and biases, the computational complexity of the algorithm shown in Code Block 7 is bounded by  $C \left[ K^2 (n_* + 1)^{2K+1} (\log(1/\delta_*) + \log(n_*)) + K^2 (n_* + 1)^{2K} \log(1/\delta_*)^2 \right]$ , as well as at most  $(n_* + 1)^{K+1}$  evaluations of  $\check{f}$ . The constructed network  $\Phi_{n_*, \delta_*}^{\check{f}}$  satisfies

$$\begin{aligned} \text{depth} \left( \Phi_{n_*, \delta_*}^{\check{f}} \right) &\leq C \left[ 1 + \log(n_*)^3 + (\log(K) + \log(n_*)) \log(1/\delta_*) + K \log(n_*) \right. \\ &\quad \left. + K \log(K) \right], \\ \text{size} \left( \Phi_{n_*, \delta_*}^{\check{f}} \right) &\leq C \left[ 1 + K^2 (n_* + 1)^K (\log(1/\delta_*) + \log(n_*)) \right]. \end{aligned}$$

```

1 def compute_dnn(k, eval_f, epsilon, alpha, delta_0, n_0):
2     # k: input dimension
3     # eval_f: function to approximate
4     # epsilon: error tolerance
5     # alpha: parameter for error tolerance
6     # delta_0: initial delta
7     # n_0: initial n
8     # returns DNN approximating eval_f with error tolerance
9     # epsilon as described in Chapter 3, depth of the DNN, size of
10    # the DNN, n, delta and the error in the L_\infty norm
11    n = n_0
12    delta = delta_0
13    state = True
14    err = 1
15    iter = 1
16    while err > epsilon:
17        var_lambda = generate_combinations(n, k)
18        err_1 = err
19        chebyshev = Chebyshev(eval_f, n, k)
20        _, coeffs = chebyshev.coefficients()
21        phi_l_d = phi_Lambda_delta(k, var_lambda, delta)
22        nn_coeffs = nn.NeuralNetwork(len(var_lambda), [], 1)
23        for i in range(len(var_lambda)):
24            nn_coeffs.weights[0][0][i] = coeffs[i]
25        phi = nn_coeffs.concatenate(phi_l_d)
26        xx = cos_nd(generate_combinations(n-1, k), n-1)
27        f_interp = chebyshev.interpolate(xx, coeffs)
28        f_realized = [phi.realize(x) for x in xx]
29        err = np.max(np.absolute(f_interp - f_realized))

```

```

28     if err > alpha*err_1:
29         state = not state
30     if state:
31         n = n+1
32     else:
33         delta = .5*delta
34     print(f'Iteration: {iter} Error: {err}, n: {n}, delta: {
delta}')
35     iter += 1
36     if state:
37         n_star = n-1
38         delta_star = delta
39     else:
40         n_star = n
41         delta_star = 2*delta
42     var_lambda = generate_combinations(n_star, k)
43     chebyshev = Chebyshev(eval_f, n_star, k)
44     _, coeffs = chebyshev.coefficients()
45     phi_l_d = phi_Lambda_delta(k, var_lambda, delta_star)
46     nn_coeffs = nn.NeuralNetwork(len(var_lambda), [], 1)
47     for i in range(len(var_lambda)):
48         nn_coeffs.weights[0][0][i] = coeffs[i]
49     phi = nn_coeffs.concatenate(phi_l_d)
50     xx = cos_nd(generate_combinations(n_star-1, k), n_star-1)
51     f_interp = chebyshev.interpolate(xx, coeffs)
52     f_realized = [phi.realize(x) for x in xx]
53     err = np.max(np.absolute(f_interp-f_realized))
54     depth, size = phi.count_depth_size()
55     print(f"Total depth: {depth}, Total size: {size}, n: {n_star},
delta: {delta_star}, error: {err}. Reached after {iter}
iterations.")
56     return phi, depth, size, n_star, delta_star, err

```

Code Block 7: Algorithm for the construction of a DNN for approximating a given function as can be found in the file *dnn\_construction.py*

*Proof.* The while loop in lines 14 – 35 of Code Block 7 is - after the first iteration - executed at most  $n_* - n_0$  times after increasing  $n$  and  $\log_2(\delta_0/\delta_*)$  times after decreasing  $\delta$ , which adds up to a maximum of  $n_* - n_0 + 1 + \log_2(\delta_0/\delta_*)$  executions. For each iteration of the while loop the  $(n+1)^K$  evaluations of  $f$  defined in line 25 are executed for  $n = n_0, \dots, n_*$ . The coefficients computed with a  $K$ -dimensional array of size  $(n+1)^K$  in line 18 have a complexity of each execution of  $CK(n+1)^K \log(n+1) = C(n+1)^K \log((n+1)^K)$ . Line 19 contains the construction of  $\Phi_{\Lambda, \delta}$ , which has computational cost  $CK^2(n+1)^K(\log(1/\delta) + \log(n))$  as follows from Theorem 2.9. The evaluation of  $\Phi_{n, \delta}^f$  in line 26 adds  $CK^2(n+1)^{2K}(\log(1/\delta) + \log(n))$  for each iteration.

The total computational complexity is bounded by  $(n_* + 1)^{K+1}$  evaluations of  $\check{f}$  in addition to  $CK^2(n_* + 1)^{2K+1}(\log(1/\delta_*) + \log(n_*)) + CK^2(n_* + 1)^{2K} \log(1/\delta_*)^2$ , with  $C > 0$  being a constant depending on  $\delta_0$  and  $n_0$ . The bounds on the network depth and size follow from Theorem 2.8 and

$$\begin{aligned} \text{depth}(\Phi_{n_*, \delta_*}^{\check{f}}) &\leq \text{depth}(((A, b))) + 1 + \text{depth}(\Phi_{\Lambda, \delta}), \\ \text{size}(\Phi_{n_*, \delta_*}^{\check{f}}) &\leq C \text{size}(((A, b))) + C \text{size}(\Phi_{\Lambda, \delta}), \end{aligned}$$

where  $(A, b)$  is the DNN holding the coefficients  $\check{f}_{k;n}$  as described above.  $\square$

It should be noted that choosing a value for  $n_0$  that is too small may result in an unwanted termination before reaching desired accuracy. This might happen if the approximation  $\check{f}$  and the DNN  $\Phi_{n, \delta}^{\check{f}}$  map to 0 for all gridpoints, even though  $f$  is not the zero function. This will be avoided by choosing  $n_0$  sufficiently large.

## 4 Numerical Results

Having implemented the DNNs as described in the previous section, the following section will present some numerical results.

### 4.1 Approximation of Univariate Functions

To start testing, the univariate function

$$f(x) = e^{-|x|}(\sin(4\pi x) + \cos(2\pi(x - 1/4))), \quad x \in [0, 1]$$

was chosen. A plot of the function can be seen in Figure 4, the following computations can be found in the file *example\_1\_1d\_func.ipynb*.

Approximating this function by classical Chebyshev approximation, as described in Theorem 2.1, shows that for this function to be approximated by even a generous  $L^\infty$ -error of  $\varepsilon \leq 10^{-1}$ , a degree of  $n \geq 20$  is needed (even for a degree of  $n = 100$  the error only decreases by the factor 10 compared to  $n = 20$ ). The resulting approximations can be seen in Figure 5. The errors can be seen in Figure 6 and Table 1<sup>10</sup>.

The DNN approximation of the function was computed using the algorithm described in Code Block 7. After the preparations described in the previous chapters, the implementation itself is simple and can be seen in Code Block 8. As starting

---

<sup>10</sup>MSE denoting the *mean squared error*, cf. [29]

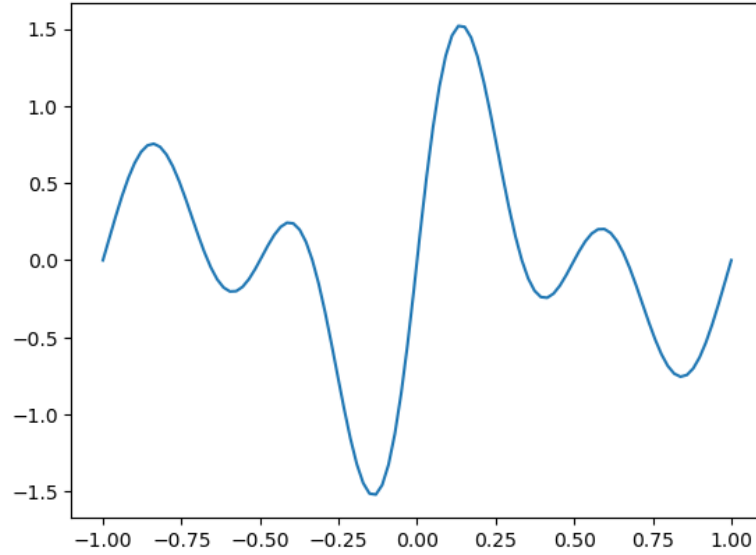


Figure 4: Plot of the function  $f(x) = e^{-|x|}(\sin(4\pi x) + \cos(2\pi(x - 1/4)))$  for  $x \in [-1, 1]$

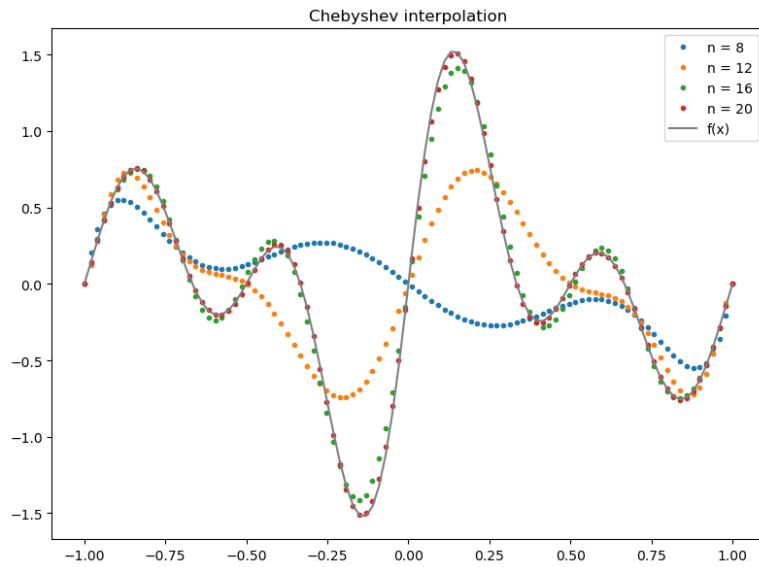


Figure 5: Chebyshev approximation of the function  $f(x) = e^{-|x|}(\sin(4\pi x) + \cos(2\pi(x - 1/4)))$  for  $x \in [-1, 1]$

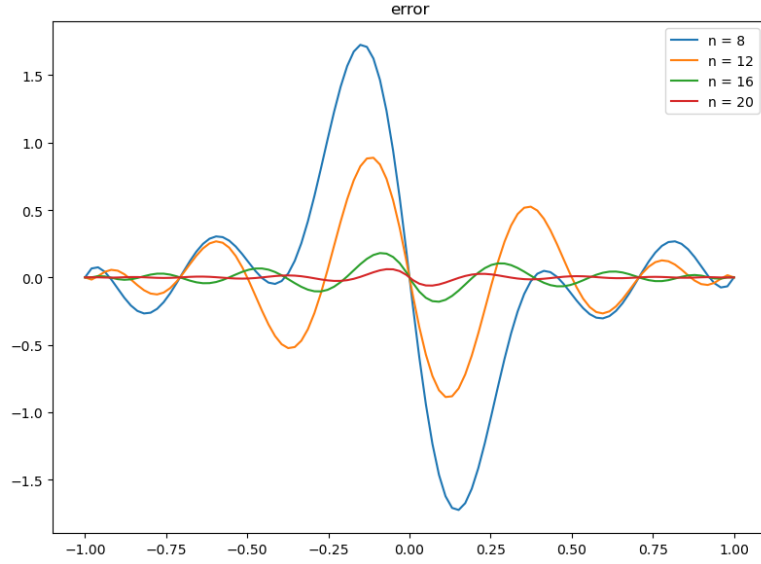


Figure 6: Error of the Chebyshev approximation of the function  $f(x) = e^{-|x|}(\sin(4\pi x) + \cos(2\pi(x - 1/4)))$  for  $x \in [-1, 1]$

Table 1: Errors with Chebyshev approximation

$n$	$L^\infty$ -error	MSE
8	1.7249	0.5103
12	0.8873	0.1422
16	0.1804	0.0047
20	0.0607	$3.66 \cdot 10^{-4}$

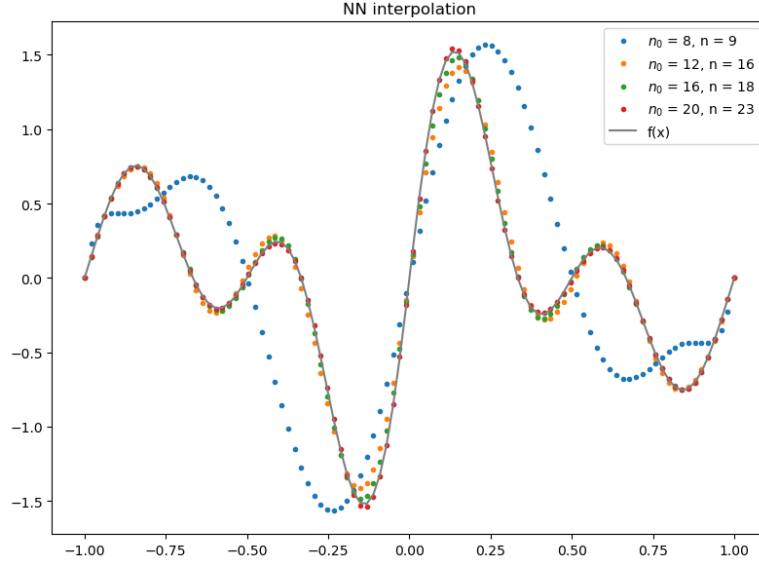


Figure 7: Approximation of the function  $f(x) = e^{-|x|}(\sin(4\pi x) + \cos(2\pi(x - 1/4)))$  for  $x \in [-1, 1]$  by the constructed ReLU DNN  $\Phi_{n_*, \delta_*}^f$

parameters  $n_0$  the degrees of the Chebyshev approximations were chosen, while  $\delta_0 = \alpha = 0.5$  was chosen for all of the DNNs. By design the algorithm will then increase  $n$  or decrease  $\delta$  until the desired accuracy is reached. The results can be seen in Figure 7 and the errors in Figure 8.

```

1 from dnn_construction import compute_dnn
2 for n_0 in [8, 12, 16, 20]:
3     net = compute_dnn(1, f, 1e-09, 0.5, 0.5, n_0)
4     y_net = np.array([net[0].realize(xi) for xi in x])
    
```

Code Block 8: Implementing the construction of the DNNs as can be found in the file *example\_1.1d\_func.ipynb*

For  $n_0 = 8$  and  $\delta_0 = 0.5$  the algorithm terminated after 8 iterations with  $n_* = 9$  and  $\delta_* = 0.015625$ . The resulting DNN has a depth of 138 and a size of 32560. The  $L^\infty$ -error of the approximation is computed as  $\approx 6 \cdot 10^{-10}$ . As is immediately obvious, the computed error is based on too few evaluated points and the constructed DNN approximates a corrupted interpolation of the target function  $f$ , since the actual error is much higher. For higher  $n_0$  the target function is approximated a lot more closely, as can be seen in table 2.

From this simple example a few remarks can be made:

First it is noteworthy that the depth of the constructed DNNs will increase in a

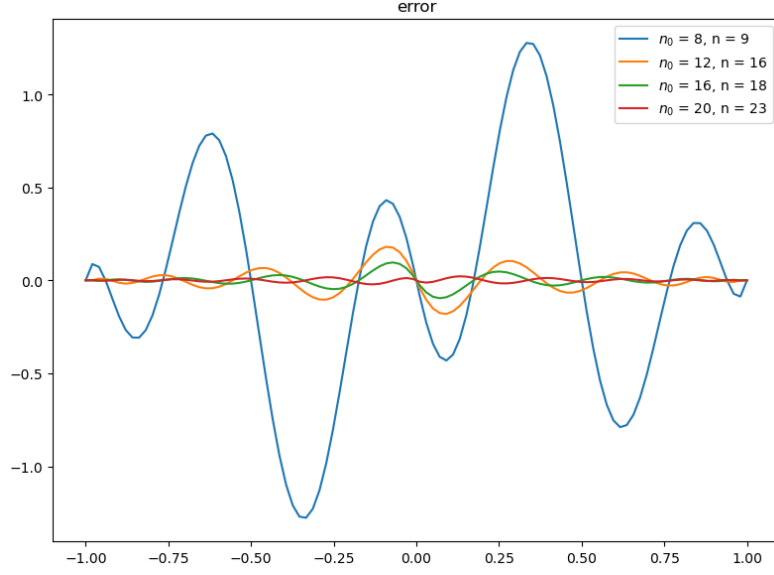


Figure 8: Error of the approximation of the function  $f(x) = e^{-|x|}(\sin(4\pi x) + \cos(2\pi(x - 1/4)))$  for  $x \in [-1, 1]$  by the constructed ReLU DNN  $\Phi_{n_*, \delta_*}^{\tilde{f}}$

Table 2: Results with the constructed DNN							
$n_0$	$n_*$	$\delta_*$	depth	size	$L^\infty$ -error	MSE	iterations
8	9	0.015625	138	32560	1.2768	0.3599	8
12	16	0.03125	138	32748	0.1804	0.0048	10
16	18	0.03125	212	76610	0.0962	0.0011	8
20	23	0.03125	212	76746	0.0218	$7.72 \cdot 10^{-5}$	9

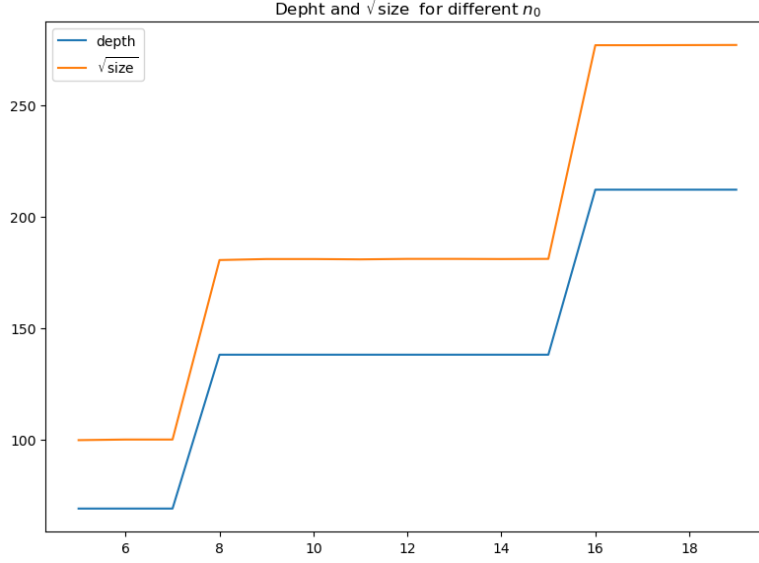


Figure 9: Behaviour of depth and size of the constructed DNN for different starting parameters  $n_0$

stepwise behaviour for increasing  $n$ , with an increase in depth for every time  $n \geq 2^k$  for some new  $k \in \mathbb{N}$ . This is due to the fact, that the multiplication of different Chebyshev polynomials as DNNs is computed by the function  $R^\ell$  in the proof of Proposition 2.6, which works by repeated pairwise multiplication of  $2^k$  many inputs for some  $k$  and artificially increasing the length of the input by adding ones if necessary. The size will increase steadily with growing  $n$  with drastic increases in size, if the depth increases. This again makes sense, since an increase in depth will add new weights and biases that add to the size. This behaviour can be seen in Figure 9.

Secondly, as mentioned,  $n_0$  has to be chosen large enough, so that enough points are evaluated to approximate the target function. Otherwise an incorrect function might be approximated with high accuracy, deeming the constructed DNN useless. This can be seen in Figure 7 for  $n_0 = 8$ .

Lastly it is remarkable that even for a one-dimensional problem the depth and size of the constructed DNNs are quite large. This is due to the fact that the constructed DNNs are not optimized in any way, but are constructed purely by the algorithm described in Code Block 7 which consists of repeated concatenations and parallelizations of DNNs. Although, as also shown in the theory, the accuracy can be made arbitrarily high, the depth and size of the constructed DNNs might



pose problems in actual implementation. A comparison with classical methods for approximating functions by trained DNNs is given in the next section.

#### 4.1.1 Comparison with trained DNNs

To compare the constructed DNNs with trained DNNs, the same function as in the previous section was approximated by a trained DNN. The following computations can be found in the file *example\_1\_1d\_trained.ipynb*. The trained DNNs were constructed with *PyTorch* [22]. In this chapter the constructed DNN will be the construction with  $n_0 = 20$ , unless stated otherwise.

In a first step the function was approximated by a trained DNN with 3 hidden layers with 1024 neurons each. These numbers were chosen by experiment for practical reasons<sup>11</sup>. For the explicit construction of the DNN a simple class was written, as can be seen in Code Block 9. The DNN was trained for 15000 epochs with a learning rate of  $10^{-6}$ . The starting values of the weights and biases were chosen randomly. The optimizer used was *Adam* [13]. MSE was used as the loss function. The resulting DNN has a depth of 3 and a size of 2098177.

```
1 class Regressor(torch.nn.Module):
2     def __init__(self, input_dim, output_dim, hidden_layers,
3         hidden_neurons):
4         super().__init__()
5         self.flatten = torch.nn.Flatten()
6         self.regressor_stack = torch.nn.Sequential()
7         self.regressor_stack.add_module("input_layer", torch.nn.
8             Linear(input_dim, hidden_neurons))
9         self.regressor_stack.add_module("relu", torch.nn.ReLU())
10        for hidden_layer in range(hidden_layers):
11            self.regressor_stack.add_module(f"hidden_layer_{
12                hidden_layer}", torch.nn.Linear(hidden_neurons, hidden_neurons)
13            )
14            self.regressor_stack.add_module(f"relu_{hidden_layer}"
15                , torch.nn.ReLU())
```

---

<sup>11</sup>For a comparable size of the networks, one would have to choose about 194 neurons per layer for three hidden layers. Then the network would have a size of about  $5 \cdot 194 + 2 \cdot 194^2 + 1 = 76243$ , similar to the constructed DNN for  $n_0 = 20$ , where the size was 76476 as shown in Table 2. For having depth and size comparable, the depth would have to be chosen as 212 with 19 neurons per hidden layer. While this will work, the training of the network will take about 20 times as long. Since the constructed DNNs will by construction have plenty of 0 entries in the weights and biases and the trained DNNs will not and since the dimensions are not equal in every layer for the trained DNNs, the architecture is not really comparable at this point. A more comparable setting will be discussed next. For this reason the number of neurons per layer was chosen to be 1024 and the depth was chosen to be 3.

```
12         self.regressor_stack.add_module("output_layer", torch.nn.  
Linear(hidden_neurons, output_dim))  
13  
14     def forward(self, x):  
15         x = self.flatten(x)  
16         output = self.regressor_stack(x)  
17         return output
```

Code Block 9: Regressor class used to construct the trained DNNs as can be found in the file *example\_1\_1d\_trained.ipynb*

As can be seen in Figure 10, after approximately 10000 epochs the training loss is almost at 0 and does not decrease significantly further - the model can be considered fully trained. The validation loss is at about 0.0036 at the end of training<sup>12</sup>. The resulting approximation can be seen in Figure 11.

Having successfully implemented a trained DNN to approximate the function that was approximated by a constructed DNN in the previous chapter, a way to compare the two approaches is needed. For this purpose the architecture of the trained DNN was changed to match the architecture of the constructed DNN and the weights and biases from the constructed DNN were used as starting values for the trained DNN. For this purpose the Regressor class was rewritten to match the syntax of the DNNs constructed in the previous chapters. This can be seen in Code Block 10.

```
1 class NNRegressor(torch.nn.Module):  
2     def __init__(self, input_dim, output_dim, hidden_dims):  
3         super().__init__()  
4         self.flatten = torch.nn.Flatten()  
5         self.regressor_stack = torch.nn.Sequential()  
6         self.regressor_stack.add_module("input_layer", torch.nn.  
Linear(input_dim, hidden_dims[0]))  
7         self.regressor_stack.add_module("relu", torch.nn.ReLU())  
8         for i in range(1, len(hidden_dims)):  
9             self.regressor_stack.add_module(f"hidden_layer_{i}",  
torch.nn.Linear(hidden_dims[i-1], hidden_dims[i]))  
10            self.regressor_stack.add_module(f"relu_{i}", torch.nn.  
ReLU())  
11  
12            self.regressor_stack.add_module("output_layer", torch.nn.  
Linear(hidden_dims[-1], output_dim))  
13  
14    def forward(self, x):  
15        x = self.flatten(x)
```

---

<sup>12</sup>This might obviously change from run to run, but should remain in the same order of magnitude.

```
16         output = self.regressor_stack(x)
17         return output
```

Code Block 10: NNRegressor class used to construct the trained DNNs with the same architecture as the constructed DNNs as can be found in the file *example\_1-1d\_trained.ipynb*

This manipulated and yet untrained DNN can then be made to train as before. One could assume that the training would need fewer epochs, since the DNN is already a better approximation for the target function than if it has random initial weights and biases. For the first test of this idea the DNN with  $n_0 = 20$  was used and the DNN trained with a low learning rate of  $10^{-9}$  and 1000 epochs. The learning rate was chosen low, since the expectation is that not a lot of changes have to be made by the optimizing algorithm. Indeed, as can be seen in Figure 12, the loss function starts out low and quickly reaches a minimum. As can be seen in Figure 13, the resulting approximation is again very close to the target function.

A setup that might be more useful in future applications is to use a constructed DNN with a low  $n_0$  and high values for  $\delta_0$  and  $\varepsilon$ , that should be computable at low cost, to generate starting values for DNN training, that outperform random starting values and reduce the training time. For this purpose the constructed DNN with  $n_0 = 5$ ,  $\delta_0 = 0.9$  and  $\varepsilon = 0.1$  was used to generate starting values for the trained DNN. The resulting DNN can be seen in Figure 14. It approximates  $f$  poorly, but not with a constant 0 function as might happen for lower values of  $n_0$ . The resulting DNN was then trained for 10000 epochs with a learning rate of  $10^{-5}$  and the same optimizer and loss function as before. The training loss can be seen in Figure 15. The resulting approximation can be seen in Figure 16.

It is very obvious, that the the behaviour of the loss function is highly unusual and unexpected. While the sudden drop in the beginning might be expected, the peak in the loss function after about 6500 epochs is not. The resulting approximation is - while better than the approximation by the constructed DNN - still far from the expected high-accuracy approximation usually obtained from trained DNNs. It should furthermore be noted, that the same setup with different learning rates (higher and lower) would fail completely and the loss would either constantly rise unexpectedly or compute to *NaN* values. The convergence will also fail completely using *SGD* as an optimizer instead of *Adam*. This behaviour invites further research and might be due to the fact that the weights and biases of the constructed DNN are not optimized for training (e.g. they contain many zeros) and might therefore be unsuitable. However, since the constructed DNN with  $n_0 = 20$  was successfully used to generate starting values for training, this is not the only possible explanation.

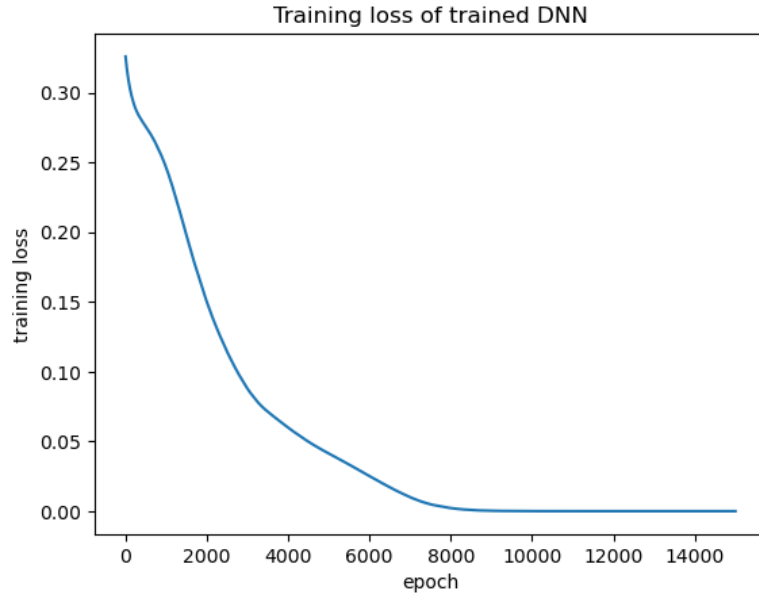


Figure 10: Training loss of the DNN trained to approximate  $f$

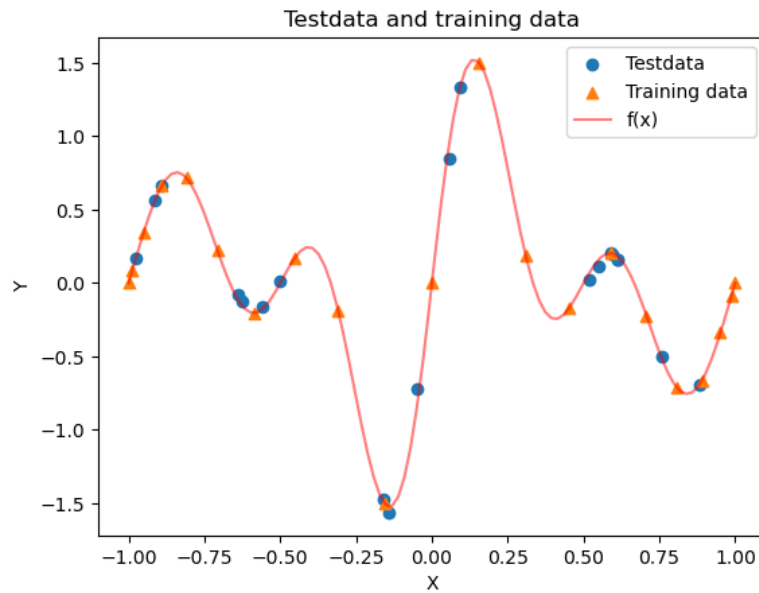


Figure 11: Testdata used for approximation and the resulting approximation by the trained DNN

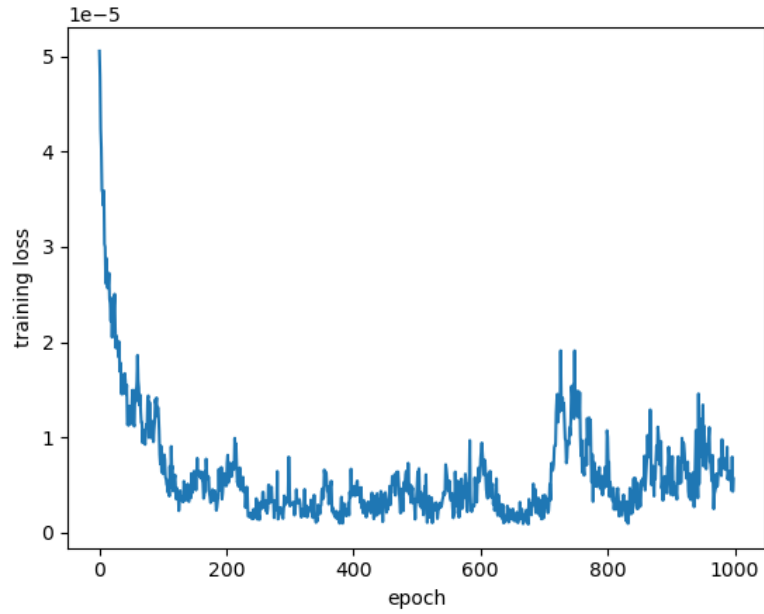


Figure 12: Training loss of the manipulated trained DNN with weights and biases inserted from constructed DNN with  $n_0 = 20$

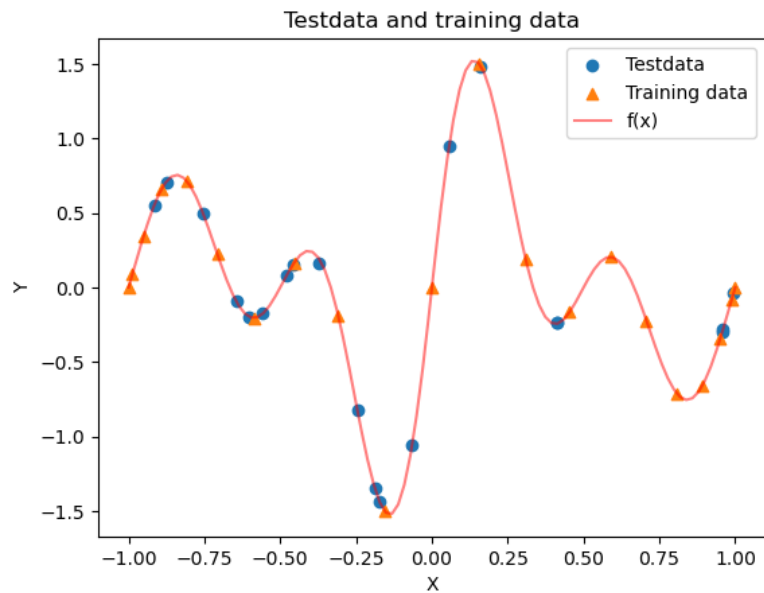


Figure 13: Approximation by manipulated trained DNN with weights and biases inserted from constructed DNN with  $n_0 = 20$

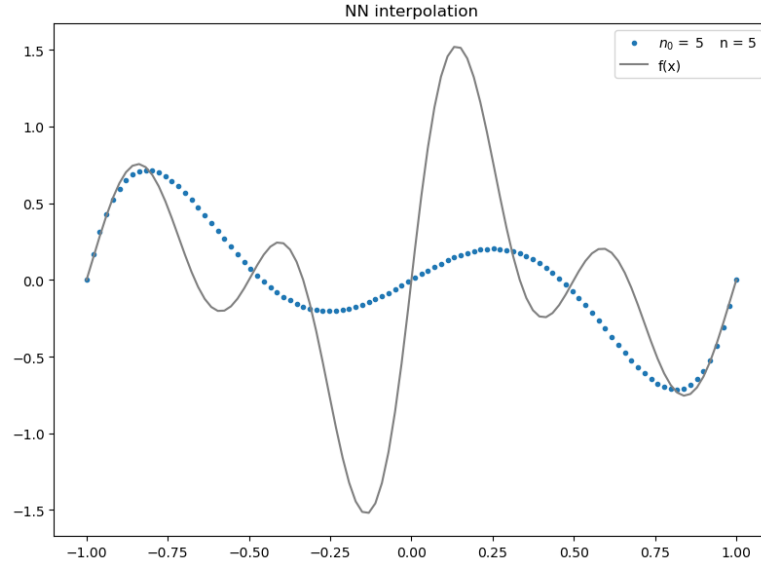


Figure 14: Approximation of  $f$  by constructed DNN with  $n_0 = 5$

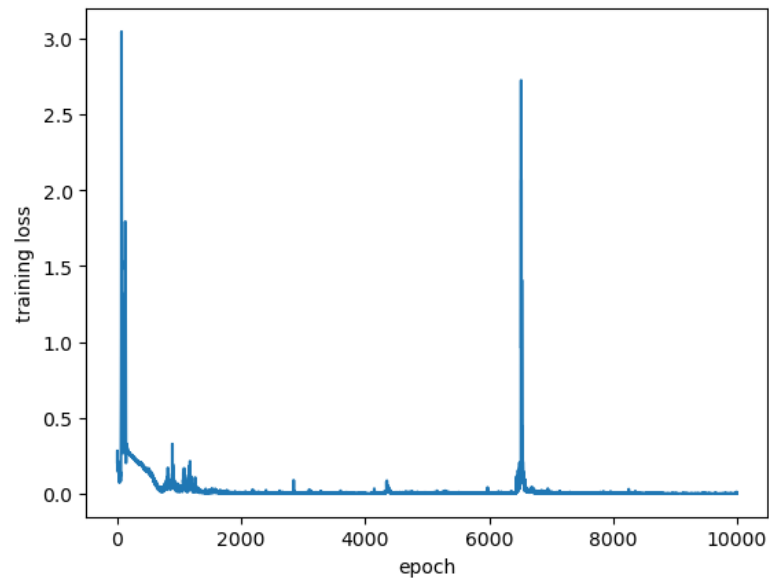


Figure 15: Training loss of the manipulated trained DNN with weights and biases inserted from constructed DNN with  $n_0 = 5$

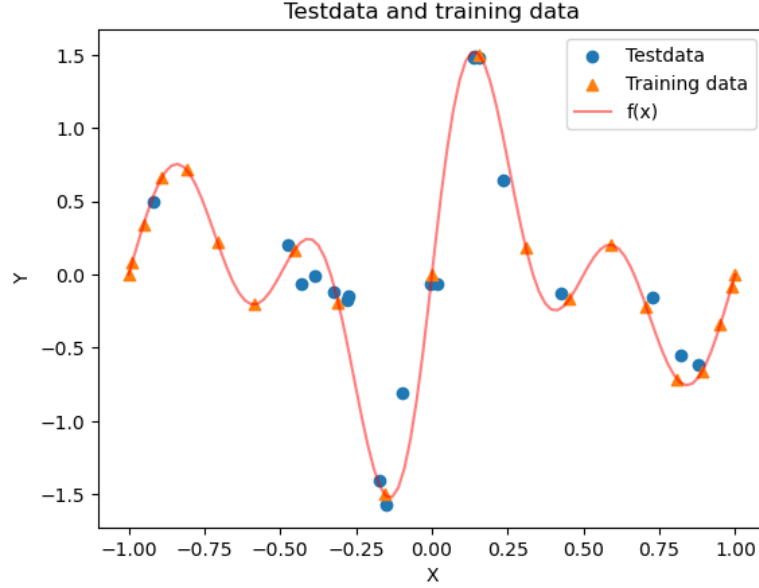


Figure 16: Approximation by manipulated trained DNN with weights and biases inserted from constructed DNN with  $n_0 = 5$

## 4.2 Approximation of Multivariate Functions

After having tested the constructed DNNs in the one-dimensional case, the next step is to test the DNNs in higher dimensions. For this purpose the function

$$f(x, y) = e^{-|x+y|}(\sin(\pi x/4) + \cos(2\pi(y - 1/4))), \quad x, y \in [-1, 1]^2$$

was chosen. A plot of the function can be seen in Figure 17. The following computations can be found in the file *example\_2-2d\_func.ipynb*.

As in the one-dimensional case, the function was approximated by classical Chebyshev approximation. For this purpose the function was approximated by a tensor product of Chebyshev polynomials in each dimension. As degrees of approximation  $n = 12$  and  $n = 20$  were chosen for testing. The resulting approximations can be seen in Figure 18 and Figure 19. The  $L^\infty$ -error on all  $50 \times 50$  interpolation points was computed as 0.8685 for  $n = 12$  and 0.0416 for  $n = 20$ .

Analogously to the one-dimensional case, the constructed DNN was used to approximate the function. For this purpose the algorithm described in Code Block 7 was used. As starting parameters  $n_0$  the degrees of the Chebyshev approximations were chosen, while  $\delta_0 = \alpha = 0.5$  was chosen for all of the DNNs. The results can be seen in Figure 20 and the errors in Figure 21.

The  $L^\infty$ -error on all  $50 \times 50$  interpolation points was quite similar to the error of

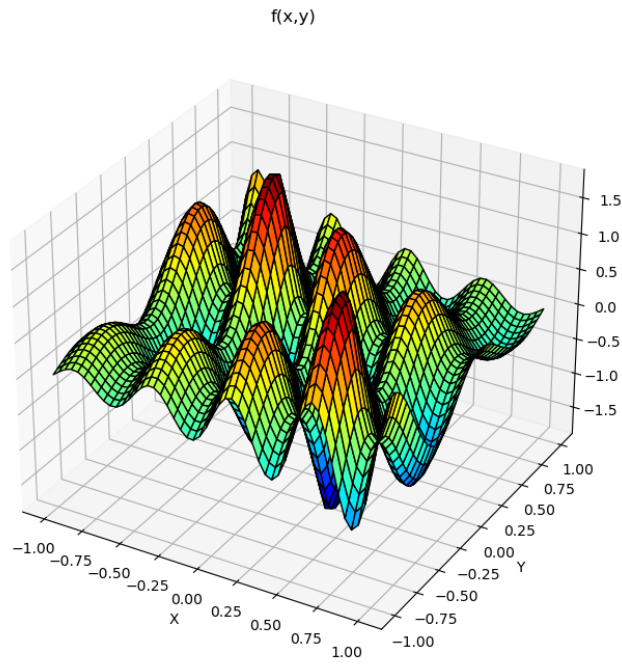


Figure 17: Plot of the function  $f(x,y) = e^{-|x+y|}(\sin(\pi x/4) + \cos(2\pi(y - 1/4)))$

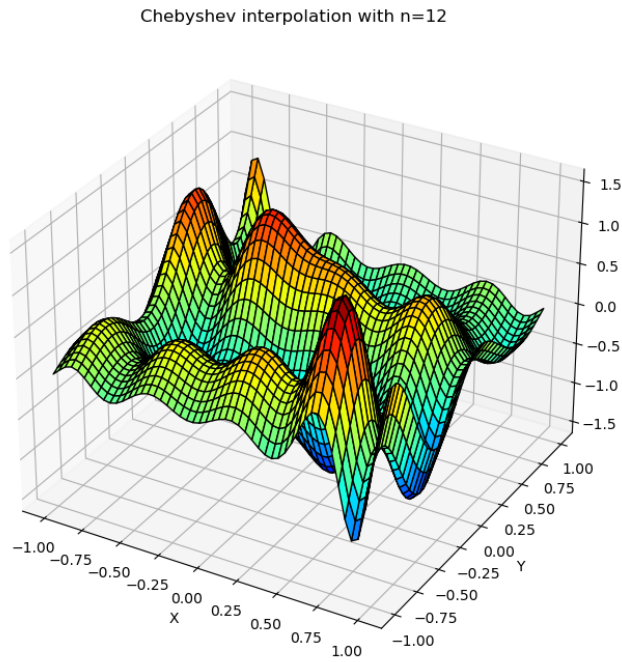


Figure 18: Chebyshev approximation of  $f$  with  $n = 12$



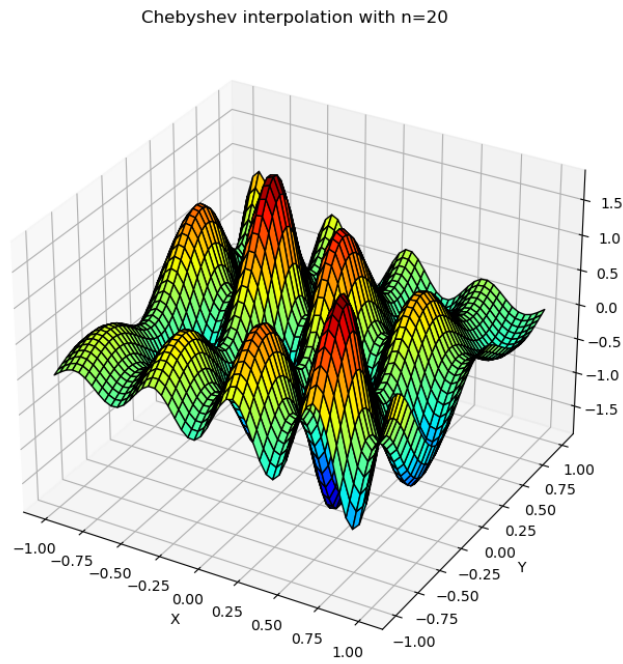


Figure 19: Chebyshev approximation of  $f$  with  $n = 20$

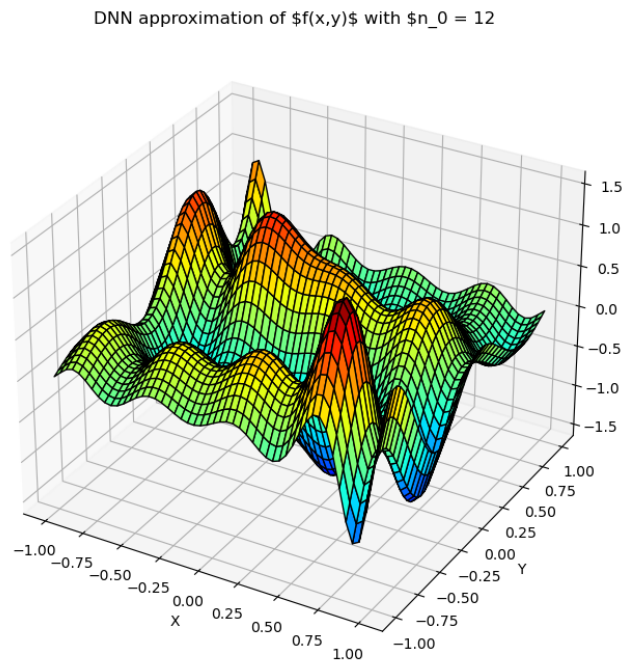


Figure 20: Approximation of  $f$  with constructed DNN and  $n_0 = 12$

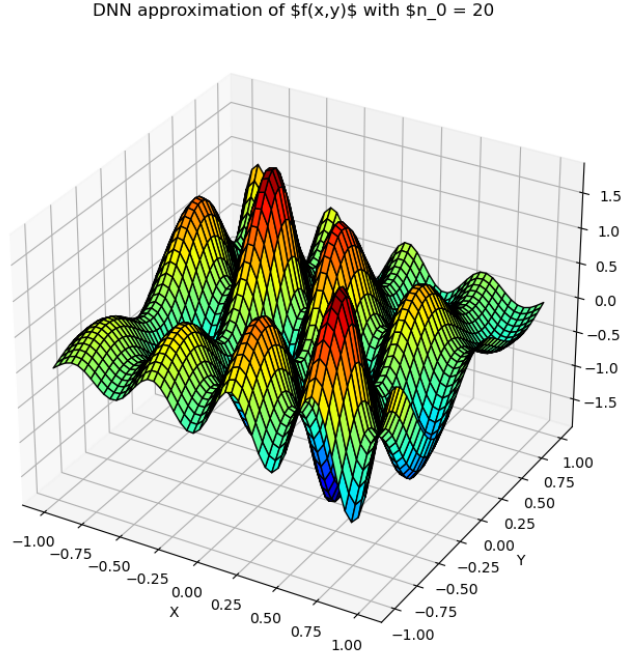


Figure 21: Approximation of  $f$  with constructed DNN and  $n_0 = 20$

Table 3: Results with the constructed DNN

$n_0$	$n_*$	$\delta_*$	depth	size	$L^\infty$ -error	iterations
12	12	0.5	141	184154	0.8825	2
20	20	0.5	218	495527	0.0899	2

the Chebyshev approximation and can be found in Table 3, as well as the depth and size of the constructed DNNs. Both times the algorithm terminated after two iterations with  $n_* = 12$  and  $\delta_* = 0.5$  for  $n_0 = 12$  and  $n_* = 20$  and  $\delta_* = 0.5$  for  $n_0 = 20$ . It should be noted that the depth and in particular the size of both DNNs is quite large and increases further with higher dimension. This has to be taken into account when using the constructed DNNs in practice. Injecting the weights and biases of these DNNs as starting values into the training of a DNN is limited by this fact and might not be possible for higher dimensions without a lot of computational power.

## 5 Comparisons

After having presented the theory and the actual implementation of the explicit construction of ReLU DNNs that approximate given functions using the emulation

of multivariate Chebyshev polynomials, a small selection of different results from other publications is presented to show the rich potential of studying various constructive approaches to understanding the inner workings of DNNs. It should be noted that this comparison will focus solely on those approaches that share with [10] the idea of direct construction of a DNN and not on those that focus on other aspects of DNN research, such as loss function minimization algorithms for DNN training or the impact of depth.

[21] presents a similar idea to the one discussed in this thesis. The paper focuses on functions  $f \in W^{s,p}([-1, 1])$ ,  $p \in [1, \infty]$ ,  $s \geq 1$  that are approximated by a ReLU DNN-emulation of classical polynomial interpolation. One of the main results states:

**Proposition 5.1.** For each  $n \in \mathbb{N}_0$  and each polynomial  $v \in \mathbb{P}_n([-1, 1])$ , such that  $v(x) = \sum_{\ell=0}^n \tilde{v}_\ell x^\ell$ , for all  $x \in [-1, 1]$  with  $C_0 := \sum_{\ell=2}^n |\tilde{v}_\ell|$ , there exist  $NNs \{\Phi_\delta^v\}_{\delta \in (0,1)}$  with input dimension one and output dimension one which satisfy

$$\begin{aligned} \|v - R(\Phi_\delta^v)\|_{W^{1,\infty}(I)} &\leq \delta \\ R(\Phi_\delta^v)(0) &= v(0) \\ \text{depth}(\Phi_\delta^v) &\leq C_L (1 + \log_2(n)) \log_2(C_0/\delta) + \frac{1}{3} C_L (\log_2(n))^3 \\ &\quad + C (1 + \log_2(n))^2 \\ \text{size}(\Phi_\delta^v) &\leq 2C_M n \log_2(C_0/\delta) + 4C_M n \log_2(n) \\ &\quad + 4C_L (1 + \log_2(n)) \log_2(C_0/\delta) + C(1 + n) \end{aligned}$$

if  $C_0 > \delta$ . If  $C_0 \leq \delta$  the same estimates hold, but with  $C_0$  replaced by  $2\delta$ .

$C_m$  and  $C_L$  are constants from preliminary results. A direct comparison to Lemma 2.7 shows that the bounds on depth and size are smaller for the approximation of Chebyshev polynomials (choosing the maximum of all constants as a constant for both approaches). Based on this result, [21] goes on to show that DNNs can emulate a high-order finite element method (FEM) on general partitions of a bounded interval. It is shown, that concerning an approximation theoretical viewpoint, DNNs perform as well as the best available FEM.

A more efficient approximation with respect to the  $W^{1,\infty}$ -norm and with a less strict condition on smoothness is presented in [20] by the same authors as [10]. It presents the following theorem by a constructive proof:

**Proposition 5.2.** Fix  $d \in \mathbb{N}$  and let  $\rho = (\rho_j)_{j=1}^d \in (1, \infty)^d$ . Assume that  $u : [-1, 1]^d \rightarrow \mathbb{R}$  admits a holomorphic extension to  $\mathcal{E}_\rho$ .

Then, there exist constants  $\beta' = \beta'(\rho, d) > 0$  and  $C = C(u, \rho, d) > 0$ , and for every  $\mathcal{N} \in \mathbb{N}$  there exists a ReLU DNN  $\tilde{u}_{\mathcal{N}} : [-1, 1]^d \rightarrow \mathbb{R}$  satisfying

$$\text{size}(\tilde{u}_{\mathcal{N}}) \leq \mathcal{N}, \quad \text{depth}(\tilde{u}_{\mathcal{N}}) \leq C\mathcal{N}^{\frac{1}{d+1}} \log_2(\mathcal{N})$$

and the error bound

$$\|u(\cdot) - \tilde{u}_{\mathcal{N}}(\cdot)\|_{W^{1,\infty}([-1,1]^d)} \leq C \exp\left(-\beta' \mathcal{N} \frac{1}{d+1}\right)$$

The constructed DNNs presented here suffer from a more harsh trade-off between DNN architecture (i.e. size and depth) and optimizing the Lebesgue constants, than the Chebyshev-based constructions in this thesis and in [10].

In [17] the concept of constructing ReLU DNNs is used to show that these networks can approximate smooth functions  $f \in C^s([0, 1]^d)$  with a nearly optimal approximation error. The main result in the paper is the following:

**Proposition 5.3.** Given a smooth function  $f \in C^s([0, 1]^d)$  with  $s \in \mathbb{N}^+$ , for any  $N, L \in \mathbb{N}^+$ , there exists a function  $\phi$  implemented by a ReLU FNN with width  $C_1(N+2)\log_2(8N)$  and depth  $C_2(L+2)\log_2(4L) + 2d$  such that

$$\|\phi - f\|_{L^\infty([0,1]^d)} \leq C_3 \|f\|_{C^s([0,1]^d)} N^{-2s/d} L^{-2s/d},$$

where  $C_1 = 17s^{d+1}3^d d$ ,  $C_2 = 18s^2$ , and  $C_3 = 85(s+1)^d 8^s$  and where the norm  $\|\cdot\|_{C^s([0,1]^d)}$  is defined by

$$\|f\|_{C^s([0,1]^d)} := \max \left\{ \|\partial^\alpha f\|_{L^\infty([0,1]^d)} : \|\alpha\|_1 \leq s, \alpha \in \mathbb{N}^d \right\}$$

for any  $f \in C^s([0, 1]^d)$ .

The paper [7] is an example for the combination of the idea of explicit construction of DNNs with the classical approach of training DNNs by loss function minimization through algorithms such as Stochastic Gradient Descent (SGD). Based on the same construction of  $g$ ,  $g_m$  and  $f_m$  as in 2.5 this thesis, based on the results of [32], the authors show that the performance of a DNN can be enhanced by subsequently adding new neurons to the network and minimizing the loss in each step instead of minimizing the loss of the whole network at once, especially since they also show how SGD fails to converge to high relative accuracy.

## 6 Conclusion

### 6.1 Summary

The main results of this thesis are twofold: Firstly, in Chapter 2, the two main theorems were proven, the first of which, Theorem 2.8 shows the existence, error bound and bound on depth and size for DNNs that emulate multivariate Chebyshev polynomials. This was achieved by showing the existence of DNNs that are able to perform multiplication of two and of an arbitrary number of inputs. This was then used to emulate Chebyshev polynomials of the first kind by DNNs and this could then again be applied to achieve the multivariate case. The DNNs were, as all DNNs in this thesis, constructive in nature and therefore allowed for a traceable implementation. It should be mentioned that the error bounds and the bounds on depth and size for the fundamental Lemma 2.7 were not published before and have been proven by the author of this thesis.

From this the second main theorem 2.9 was derived, which shows the existence of DNNs that approximate a large set of functions by combining the first main result with the theory on Chebyshev approximation shown in Chapter 2.1. The error bound for these DNNs was presented in the  $L^\infty$ -norm and the  $W^{1,\infty}$ -norm. The DNNs were again constructive and therefore traceable.

The second main achievement of this thesis was the implementation of the described DNNs, as can be seen in 3 and the comparison to trained DNNs more commonly used in practical application. Since the results were, while constructive, purely theoretical and since one of the main goals of this work was to gain a deeper understanding of the construction of these DNNs, the implementation was done from scratch. The implementation showed, as can be seen in Chapter 4, the approximation by the constructed DNNs works and furthermore provided some expected and some unexpected results, such as the failure of convergence of trainable DNNs manipulated by implementing the weights and biases of constructed DNNs that are comparably small in depth and size, but approximate the target function poorly.

### 6.2 Discussion and Outlook

The interesting behaviour of the loss function of DNNs set to be trained with manipulated initial values described in Chapter 4.1.1 invites further investigation. The impact of depth and size of the manipulated DNNs on the convergence of the training process could be examined, as well as the impact of the manipulated initial values (mainly the many zeros) on the convergence of the training process.

The implementation of the DNNs in this thesis seemed somewhat inefficient resulting in large depth and size of the DNNs. A promising approach to improve the efficiency might be a hybrid method of training DNNs to achieve the tasks of multiplication and combining them in the constructive way demonstrated to construct the function. Another interesting topic for further research could be the question, if for the same accuracy the bounds on depth and size could be improved by constructing DNNs achieving the same task as the DNNs in this thesis, while using a different activation function than ReLU.

Lastly, Chebyshev polynomials are used in a variety of fields of mathematics and physics, such as approximation theory, numerical analysis, differential equations, signal processing and many more [[26],[29],[30]]. This offers a wide range of possible applications for the DNNs constructed in this thesis.

Gaining a deeper understanding of the inner workings of DNNs will remain a challenge for the years to come. Approaching this challenge by a purely constructive method can provide insights into the capability and approximative power of DNNs. Construction of DNNs that approximate given functions as well as trained DNNs (although with a more complex architecture) has been achieved by the methods presented in this thesis.

## Literature

- [1] P.L. Chebyshev, A.A. Markov, and N. Sonin. *Oeuvres*. Bd. 1. Chelsea Publishing Company, 1962.
- [2] C. W. Clenshaw. “A note on the summation of Chebyshev series”. In: *Mathematics of Computation* 9 (1955), pp. 118–120.
- [3] C.W. Clenshaw and A.R. Curtis. “A method for numerical integration on an automatic computer.” In: *Numerische Mathematik* 2 (1960), pp. 197–205. URL: <http://eudml.org/doc/131456>.
- [4] I. Daubechies et al. *Nonlinear Approximation and (Deep) ReLU Networks*. 2019. arXiv: 1905.02199 [cs.LG].
- [5] Olivier Delalleau and Yoshua Bengio. “Shallow vs. Deep Sum-Product Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011.
- [6] Dennis Elbrächter et al. “DNN Expression Rate Analysis of High-Dimensional PDEs: Application to Option Pricing”. In: *Constructive Approximation* 55.1 (2021), pp. 3–71. DOI: 10.1007/s00365-021-09541-6. URL: <https://doi.org/10.1007/s00365-021-09541-6>.
- [7] Daria Fokina and Ivan Oseledets. *Growing axons: greedy learning of neural networks with application to function approximation*. 2020. arXiv: 1910.12686 [cs.LG].
- [8] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [9] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [10] Lukas Herrmann, Joost Opschoor, and Christoph Schwab. “Constructive Deep ReLU Neural Network Approximation”. In: *Journal of Scientific Computing* 90 (Feb. 2022). DOI: 10.1007/s10915-021-01718-2.
- [11] Lukas Herrmann, Christoph Schwab, and Jakob Zech. “Deep neural network expression of posterior expectations in Bayesian PDE inversion\*”. In: *Inverse Problems* 36.12 (2020), p. 125011. DOI: 10.1088/1361-6420/abaf64. URL: <https://dx.doi.org/10.1088/1361-6420/abaf64>.
- [12] Arnulf Jentzen, Benno Kuckuck, and Philippe von Wurstemberger. *Lecture notes for the lecture: Mathematical Introduction to Deep Learning*. Summer semester 2021.

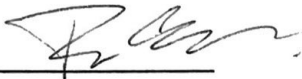
- 
- [13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].
  - [14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf).
  - [15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: <https://doi.org/10.1038/nature14539>.
  - [16] Elliott H. Lieb, Robert Seiringer, and Jakob Yngvason. *Poincare Inequalities in Punctured Domains*. 2004. arXiv: math/0205088 [math.FA].
  - [17] Jianfeng Lu et al. “Deep Network Approximation for Smooth Functions”. In: *SIAM Journal on Mathematical Analysis* 53.5 (2021), pp. 5465–5506. DOI: 10.1137/20m134695x. URL: <https://doi.org/10.1137/20m134695x>.
  - [18] Hrushikesh Mhaskar and Tomaso Poggio. *Deep vs. shallow networks : An approximation theory perspective*. 2016. arXiv: 1608.03287 [cs.LG].
  - [19] Mario Ohlberger. *Lecture notes for the lecture: Einführung in die Numerische Mathematik*. Summer semester 2010.
  - [20] Joost Opschoor, Christoph Schwab, and Jakob Zech. “Exponential ReLU DNN Expression of Holomorphic Maps in High Dimension”. In: *Constructive Approximation* 55 (2022), pp. 1–46. DOI: 10.1007/s00365-021-09542-5.
  - [21] Joost A. A. Opschoor, Philipp C. Petersen, and Christoph Schwab. “Deep ReLU networks and high-order finite element methods”. In: *Analysis and Applications* 18.05 (2020), pp. 715–770. DOI: 10.1142/S0219530519410136. eprint: <https://doi.org/10.1142/S0219530519410136>. URL: <https://doi.org/10.1142/S0219530519410136>.
  - [22] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
  - [23] Philipp Petersen and Felix Voigtlaender. *Optimal approximation of piecewise smooth functions using deep ReLU neural networks*. 2018. arXiv: 1709.05289 [math.FA].
-



- [24] W.H. Press and S.A. Teukolsky. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, 2007. ISBN: 9780521880688.
- [25] Bhavani R. and D. Sudhakar. “Design and Implementation of Inverse Fast Fourier Transform for OFDM”. In: *International Journal of Science and Engineering Applications* 02 (July 2013), pp. 155–158. DOI: 10.7753/IJSEA0207.1002.
- [26] Theodore J. Rivlin. *The Chebyshev Polynomials*. Pure and Applied Mathematics, Wiley. John Wiley and Sons Inc, 1974. ISBN: 047172470X; 9780471724704.
- [27] Christoph Schwab and Jakob Zech. “Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ”. In: *Analysis and Applications* 17.01 (2019), pp. 19–55. DOI: 10.1142/S0219530518500203. eprint: <https://doi.org/10.1142/S0219530518500203>. URL: <https://doi.org/10.1142/S0219530518500203>.
- [28] Simon J. Smith. “Lebesgue constants in polynomial interpolation.” eng. In: *Annales Mathematicae et Informaticae* 33 (2006), pp. 109–123. URL: <http://eudml.org/doc/128810>.
- [29] Lloyd N. Trefethen. “Approximation Theory and Approximation Practice, Extended Edition”. In: Philadelphia, PA: Society for Industrial and Applied Mathematics, 2019. DOI: 10.1137/1.9781611975949. eprint: <https://epubs.siam.org/doi/pdf/10.1137/1.9781611975949>. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9781611975949>.
- [30] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, 2000. DOI: 10.1137/1.9780898719598. eprint: <https://epubs.siam.org/doi/pdf/10.1137/1.9780898719598>. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9780898719598>.
- [31] Wikipedia. *Pafnuty Chebyshev* — *Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Pafnuty%20Chebyshev&oldid=1152189073>. [Online; accessed 10-January-2023]. 2023.
- [32] Dmitry Yarotsky. *Error bounds for approximations with deep ReLU networks*. 2017. arXiv: 1610.01145 [cs.LG].
- [33] K. Zeller and H. Ehlich. “Auswertung der Normen von Interpolationsoperatoren.” In: *Mathematische Annalen* 164 (1966), pp. 105–112. URL: <http://eudml.org/doc/161389>.


### Declaration of Academic Integrity

I hereby confirm that this thesis on "Explicit Construction of Deep Neural Networks" is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

May 27<sup>th</sup>, 2023 

(date and signature of student)

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

May 27<sup>th</sup>, 2023 

(date and signature of student)