Kolmogorov-Arnold representation theorem and its connection to neural networks

Pol, H. van der

Henk van der Pol

Kolmogorov-Arnold representation theorem and its connection to neural networks

Bachelor thesis

August 15, 2021

Thesis supervisor: prof. dr. A.J. Schmidt-Hieber

Leiden University
Mathematical Institute

Contents

1 Introduction

At the start of the 20th century David Hilbert posed twenty-three problems which, Hilbert argued, would give the direction for mathematical research for the next century. The thirteenth problem deals with rewriting a solution of a seven degree polynomial. This problem was disproved midway in the 20th century by Andrej Kolmogorov and his student Vladimir Arnold with the Kolmogorov-Arnold representation theorem (KA representation). Their theorem states that any multivariate continuous function can be decomposed in additive continuous functions with one variable. The decomposed form compares to a two-hidden-layer neural network and there is a long standing debate whether there is indeed a link. A true connection between the KA representation and a two-hidden-layer neural network clarifies why neural networks can avoid the so called curse of dimensionality and why having more than one hidden layer in neural networks is so useful.

Section 2 starts with an introduction to the curse of dimensionality. Specifically, we first define the non-parametric regression model underlying many machine learning methods. We assume in this thesis that the regression function f is β-Hölder-smooth. Next, we define a good approximation rate for f which avoids the curse of dimensionality. In Section 3 we introduce shallow and deep neural networks as an approximation for the non-parametric regression model. In Section 4 we introduce the Kolmogorov-Arnold representation theorem and examine its connection to neural networks. To further strengthen the link between the KA representation and neural networks, in Section 5 an extension of the KA representation of [28] is proved on the compact domain $[a, b]^d$. This extended KA representation is given using space-filling curves. Other compact domains in $\mathbb{R}^d$ are also discussed. Our goal here is to find a similar KA representation as in [28]. Finally, in Section 6, we construct a deep ReLU network which mimics the KA representation from Section 5 to approximate a β-Hölder-smooth function f. We then apply stochastic gradient descent to this deep ReLU network and analyze the expected approximation rate with respect to the order of parameters when f are the p-norms.

2 Machine learning and the curse of dimensionality

To start, we introduce the standard regression model.

Definition 2.1 (Regression model)**.** *Let $n, d \geq 1$ be integers. Given pairs $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$ which are independent and identically distributed (i.i.d.) for each $i = 1, \ldots, n$, where $\mathbf{X}_i \in \mathbb{R}^d$ are the given covariates as a vector and Y_i the observed response variable. We assume that there is a function $f : \mathbb{R}^d \to \mathbb{R}$, which we call the regression function, such that*

$$Y_i = f(\mathbf{X}_i) + \varepsilon_i, \text{ with } \varepsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0,1), \text{ for all } i = 1, \ldots, n.$$

Here, $\mathcal{N}(0,1)$ is the standard normal distribution. The given pairs $(\mathbf{X}_i, Y_i)_{i=1,\ldots,n}$ are the *data* and we name n the *sample size*. The *regression function* f in this definition, is a map from the input $\mathbf{X}_i$ to a neighbourhood of Y_i from the data. In Definition 2.1, our goal is to find the regression function that provides the best fit to the data. Therefore, we look at the function space of f. We define the function space of f as the *parameter space* given by $\mathcal{F}$. As a first example of a parameter space we have the linear regression model.

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f = \mathbf{a}^\top \mathbf{x} + b, \ \mathbf{a}, \mathbf{x} \in \mathbb{R}^d, b \in \mathbb{R}\}.$$

In this function space we only need to adjust $(d+1)$ parameters to fit the data. If the parameter space has a finite amount of parameters, we call it a *parametric model*. On the contrary, if the parameter space has an infinite amount of parameters, we call it a *non-parametric model*. This last model is of interest if we want to assume as little as possible on the regression function. In this thesis we look at the parameter space of *β-Hölder-smooth* functions, defined as follows.

Definition 2.2 (β-Hölder-smoothness)**.** *Let $d \geq 1$ be an integer, $D \subseteq \mathbb{R}^d$ and set $\beta \in (0, 1]$. We call $f : D \to \mathbb{R}$, β-Hölder-smooth if there exists a constant $C \in \mathbb{R}$ such that*

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq C\|\mathbf{x} - \mathbf{y}\|_\infty^\beta \text{ for all } \mathbf{x}, \mathbf{y} \in D.$$

Where $\|\mathbf{x}\|_\infty = \sup\{|\mathbf{x}| : \mathbf{x} \in D\}$ is the sup-norm. We can also refer to this definition as *Hölder-smooth* or *β-smooth*. For β-Hölder-smooth functions the parameter space is

$$\mathcal{F} = \left\{f : \mathbb{R}^d \to \mathbb{R} \mid f \text{ is } \beta - \text{Hölder-smooth for all } \beta \in (0, 1]\right\}.$$

If f is β-smooth for $\beta = 1$, then f is Lipschitz continuous. Furthermore, any β-smooth function is continuous. Thus, this function space is non-parametric. Another example of β-smooth functions are the real vector norms.

Theorem 2.3. *Let $p \in [1, \infty)$, integer $d \geq 1$ and $f : D \to \mathbb{R}$ the p-norm $f(\mathbf{x}) = \|\cdot\|_p$ on a compact domain $D \subset \mathbb{R}^d$. Then f is β-smooth for any positive $\beta \leq 1$.*

Proof. Let $p \geq 1$. We have for some $\mathbf{x}, \mathbf{y} \in D \subset \mathbb{R}^d$ for compact D and positive $\beta \leq 1$,

$$\begin{aligned} |f(\mathbf{x}) - f(\mathbf{y})| \leq \Big|\|\mathbf{x}\|_p - \|\mathbf{y}\|_p\Big| &\leq \|\mathbf{x} - \mathbf{y}\|_p, \\ &\leq C\|\mathbf{x} - \mathbf{y}\|_\infty, \\ &= C\|\mathbf{x} - \mathbf{y}\|_\infty^{1-\beta} \cdot \|\mathbf{x} - \mathbf{y}\|_\infty^\beta, \\ &\leq \tilde{C}\|\mathbf{x} - \mathbf{y}\|_\infty^\beta. \end{aligned}$$

For suitable constants $C, \tilde{C} \in \mathbb{R}_{\geq 0}$. In the first inequality we use the reverse triangle inequality, in the second step we use the fact that all norms are equivalent on $\mathbb{R}^d$ and in the last inequality we use the notion that all norms are bounded on compact domains in $\mathbb{R}^d$. This concludes the statement. □

We can extend the regression function by viewing it as a function of a parameter vector θ, i.e. we have $Y_i = f(\mathbf{X}_i, \theta) + \varepsilon_i$ for $i = 1, \ldots, n$ given n pairs $(\mathbf{X}_i, Y_i)_{i=1,\ldots,n}$. This *parametrizes* the regression function. Our goal then is to find appropriate parameter vector $\hat{\theta}$ that approximates the true regression function of the data. Consequently we can redefine a non-parametric model in terms of parameters. The parameter space $\mathcal{F}$ is non-parametric if the number of real parameters to parametrize $f \in \mathcal{F}$ grows with the sample size n. It is therefore difficult to reconstruct the regression function in the non-parametric model. Moreover, the space of covariates d is also of influence for the number of parameters. This is called the *curse of dimensionality* [3].

Definition 2.4 (Curse of dimensionality)**.** *The sample size n needed to estimate a non-parametric regression function up to a certain precision, grows exponentially with respect to the number of dimensions.*

As a result from this curse, we need an exponential amount of parameters to parametrize $f \in \mathcal{F}$. It is known that to approximate a β-smooth function f up to an error $m^{-\beta}$, we need at least an order of m^d parameters [28]. If we want to avoid the curse of dimensionality, we need an order of parameters less than m^d to approximate f up to an error of $m^{-\beta}$. The curse of dimensionality exists since the parameter space of a non-parametric model grows exponentially with respect to the number of dimensions of the domain (see Figure 1).

In machine learning, adjusting an exponential amount of parameters is very costly with respect to the run time. We therefore wish to avoid using so many parameters. Moreover, in typical machine learning applications, the input dimension d is large. For example, if our application is classifying an e-mail between spam or not-spam, our input vector are the individual words of the e-mail which is usually a huge number.

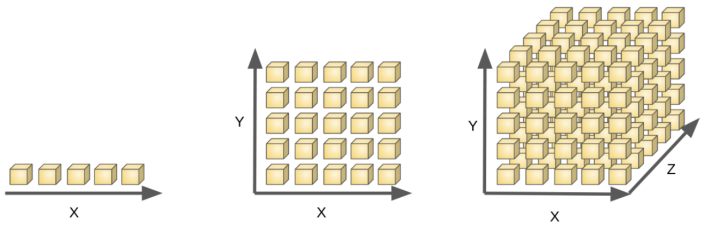

Figure 1: The curse of dimensionality visualized. As space grows exponentially for increasing number of dimensions, the function space grows exponentially as well. Therefore, the amount of parameters to parametrize the non-parametric regression function grows exponentially as well. (Source: [9])

3 Introduction to neural networks

Neural networks are at the heart of modern machine learning. The concept of neural networks is derived from observing neural activity in the human brain. For example, reading the letter Q, neurons in the brain are activated (positively) confirming we indeed read the letter Q. Another set of neurons activates whenever it reads the letter U. We can view the activity of one neuron as a function. The letter can be seen as an image $\mathbf{x} \in \mathbb{R}^d$ where d indicates the number of pixels. The neuron maps the image to the set {activated, not-activated}. This neuron function is called a *unit*. The model describing the relation of one unit is called the *perceptron*. The perceptron model is originally defined by Rosenblatt [25] as follows.

Definition 3.1 (Perceptron model)**.** *For activation function* $\sigma : \mathbb{R}^d \to \{-1, 1\}$ *with input* $\mathbf{x} \in \mathbb{R}^d$*, weights* $\mathbf{w} \in \mathbb{R}^d$ *and bias* $a \in \mathbb{R}$ *we define a unit as*

$$\sigma(\mathbf{w}^\top \mathbf{x} + a) = \operatorname{sgn}(\mathbf{w}^\top \mathbf{x} + a) = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} + a \geq 0, \\ -1 & \text{if } \mathbf{w}^\top \mathbf{x} + a < 0. \end{cases}$$

The perceptron model should return 1 if the image input is the letter Q. To increase the plane for which $\mathbf{w}^\top \mathbf{x} + a \geq 0$, we increase the weights on the pixels that make up the letter Q. In return we get a positive activation. The bias a gives more flexibility for the activation function σ. The perceptron model is not restricted to *activation function* $\sigma(x) = \operatorname{sgn}(x)$. In this paper we focus on the *Rectified Linear Unit* (ReLU) activation function. The ReLU activation function is given by $\sigma(x) = \max\{x, 0\}$ (Figure 2 right). It equals the identity function if activated and 0 if not.

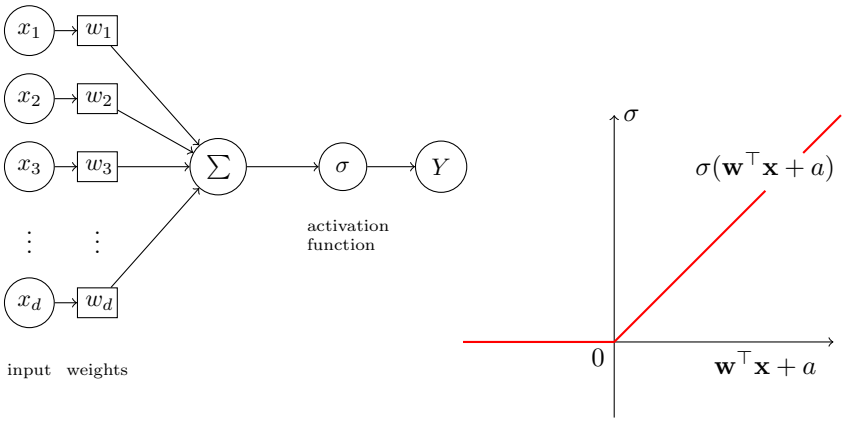

Figure 2: On the left side: The perceptron model visualized with input layer $\mathbf{x} = (x_1, \ldots, x_d)$ and on the right side: the ReLU-activation function.

We can naturally extend the number of outputs and units we use. This gives a *shallow neural network*. We define the number of units used in a shallow neural network as the *width*.

Definition 3.2 (Shallow neural network (SNN)). *A shallow neural network with input* $\mathbf{x} \in \mathbb{R}^d$*, activation function* σ*, width* $m \in \mathbb{N}$*, weights* $\mathbf{w}_j \in \mathbb{R}^d$ *and bias* $a_j, d_j \in \mathbb{R}$ *is a function* $f : \mathbb{R}^d \to \mathbb{R}$ *of the form*

$$f(\mathbf{x}) = \sum_{j=1}^{m} d_j \sigma(\mathbf{w}^\top \mathbf{x} + a_j).$$

This neural network can also be viewed as a directed acyclic graph (Figure 3). Here we have three rows of nodes. The first row is the input layer with nodes $(x_1, \ldots, x_d)$. The second row consists of m nodes which contains the computed activation function from the input layer (Figure 2 with m outputs). The third row are the results of the output layer.

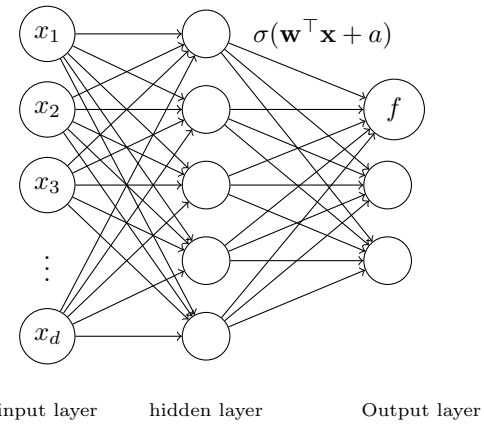

Figure 3: A shallow neural network with input $\mathbf{x} \in \mathbb{R}^d$, width $m = 5$ and output f as in Definition 3.2 for each output neuron.

We can also increase the number of hidden layers between the input layer and output layer. This gives us a *deep neural network*. To make sense of deep neural networks, we first define for each activation function in each hidden layer the *shifted activation function*. Secondly, we need a representation for the widths in each layer. This is the *network architecture*. The maximum units used in one of the hidden layers gives the width of the total deep neural network.

Definition 3.3 (Shifted activation function). *Let* $n \geq 2$ *be an integer. For a choice of activation function* σ *and shift vector* $\mathbf{v} \in \mathbb{R}^n$*, the shifted activation function* $\sigma_{\mathbf{v}} : \mathbb{R}^n \to \mathbb{R}^n$ *is defined as*

$$\sigma_{\mathbf{v}} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sigma(x_1 - v_1) \\ \vdots \\ \sigma(x_n - v_n) \end{pmatrix}.$$

Definition 3.4 (Network architecture). *A network architecture is a pair* $(L, \mathbf{p})$ *with positive integer* L *which defines the number of hidden layers and* $\mathbf{p} = (p_0, \ldots, p_{L+1}) \in \mathbb{N}^{L+2}$ *is the width vector. More precisely,* p_0 *is the width of the input layer,* p_j *the width of the* j*-th hidden layer and* p_{L+1} *the width of the output layer.*

Definition 3.5 (Deep neural network (DNN))**.** *A deep neural network or multi-layer neural network with architecture* $(L, \mathbf{p})$*, weight matrices* $W_i \in \mathrm{Mat}(p_{i+1} \times p_i, \mathbb{R})$*, shift vector* $\mathbf{v}_i \in \mathbb{R}^{p_i}$ *and shifted activation functions* $\sigma_{\mathbf{v}_i} : \mathbb{R}^{p_i} \to \mathbb{R}^{p_i}$ *is a function* $f : \mathbb{R}^{p_0} \to \mathbb{R}^{p_{L+1}}$ *of the form*

$$f(\mathbf{x}) = W_L \sigma_{\mathbf{v}_L} W_L \sigma_{\mathbf{v}_{L-1}} \dots W_1 \sigma_{\mathbf{v}_1} W_0 \mathbf{x}.$$

In the non-parametric regression model of Definition 2.1, we have $p_0 = d$ and $p_L = 1$. We can choose different activation functions σ in each hidden layer. In case that for all units in each hidden layer the ReLU activation function is chosen and $L > 1$, the network is called a *deep ReLU network*. Moreover, if the activation function overall is the identity function, then the neural network simplifies into a linear function. An example of a deep neural network is given below.

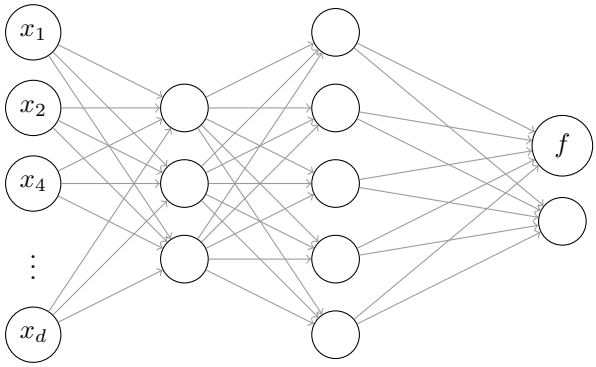

Figure 4: DNN with architecture $(2, (d, 3, 5, 2))$, input $\mathbf{x} \in \mathbb{R}^d$ and width 5. For each output unit we have $f(\mathbf{x}) = \sum_{q=1}^{5} d_q \sigma\left(\sum_{p=1}^{3} b_{pq} \sigma\left(\mathbf{w}_p^\top \mathbf{x} + a_p\right) + c_q\right)$ with parameters $\mathbf{w}_p \in \mathbb{R}^d$, $d_q, b_{pq}, a_p, c_q \in \mathbb{R}$.

We now look at neural networks as an approximation for the regression function. In a two-hidden-layer neural network with m_1 the width of the first hidden layer and m_2 the width of the second hidden layer, we have a total of $2m_2 + m_1 m_2 + (d+1)m_1$ parameters. With addition of more hidden layers, the amount of parameters needed to form a deep neural network is a polynomial function. Therefore, if we have a deep neural network with L is of smaller order than d, then the order of parameters needed to approximate a non-parametric regression function is smaller with respect to an approximation which uses an exponential amount of parameters. Moreover, if in this case we get an equal order of approximation rate, we avoid the curse of dimensionality in terms of parameters. Furthermore, if an activation function σ has the universal approximation property defined below, a shallow neural network with σ can approximate any continuous function up to an error $\varepsilon > 0$.

Definition 3.6 (Universal approximation property)**.** *Shallow neural networks with activation* σ *have the universal approximation property if for any* $\varepsilon > 0$ *and any continuous* f *on* $[0,1]^d$*, there exists an integer* $m = m(f, \varepsilon)$ *such that*

$$\inf \left\{ \|f - g\|_\infty \leq \varepsilon \,\middle|\, \begin{array}{l} f, g \text{ on } [0,1]^d, g \text{ any SNN with width } m \\ \text{and activation function } \sigma \end{array} \right\}.$$

Theorem 3.7. *Consider shallow neural networks with smooth activation function σ and σ not a polynomial. Then the universal approximation property holds.*

The proof of which can be found in [27]. Theorem 3.7 can be extended to hold for ReLU activation function [29] as well. This suggests shallow neural networks are sufficiently enough to approximate the regression function. Nevertheless, deep neural networks are widely used nowadays, outperforming shallow networks, [19]. This raises the question into why additional hidden layers provide more benefits.

4 Kolmogorov-Arnold representation theorem

In this chapter we recall the Kolmogorov-Arnold representation theorem (KA representation) and cover the main discussion of this thesis. As stated in the introduction, the Kolmogorov-Arnold representation theorem disproves Hilbert's thirteenth problem [12]: is it possible for the solution x in

$$x^7 + ax^3 + bx^2 + cx + 1 = 0,$$

seen as a three-variable function $x(a, b, c)$, to be rewritten as a composition of two-variate functions? Kolmogorov and Arnold gave a generalized negative answer to this problem in 1957 [15].

Theorem 4.1 (Kolmogorov-Arnold representation theorem)**.** *Let $d \geq 1$ be an integer and $f : [0, 1]^d \to \mathbb{R}$ be any d-variate continuous function. There exists univariate continuous functions $g_q, \psi_{p,q}$ such that*

$$f(x_1, \ldots, x_d) = \sum_{q=0}^{2d} g_q \left(\sum_{p=1}^{d} \psi_{p,q}(x_p) \right).$$

Theorem 4.1 is also called the *Superposition theorem*. If we think of each $g_q, \psi_{p,q}$ to be activation functions and for width $2d$, we have a similar representation with a shifted two-hidden-layer neural network of the form:

$$f(\mathbf{x}) = \sum_{q=1}^{m_1} d_q \sigma \left(\sum_{p=1}^{m_2} b_{pq} \sigma \left(\mathbf{w}_p^\top \mathbf{x} + a_p \right) + c_q \right), \quad \begin{array}{c} \text{with parameters } \mathbf{w}_p \in \mathbb{R}^d, \\ d_q, b_{pq}, a_p, c_q \in \mathbb{R}. \end{array}$$

This similar representation gives insight into our discussion at the end of Section 3 as it says that any d-variate function defined on the unit cube can be rewritten as a shifted two-hidden-layer neural network. This connection was made for the first time in 1987 by Hecht-Nielsen [10]. It was initially declared irrelevant by Poggio and Girosi in 1989 [23] for two reasons. First, the *outer function* g_q depends on f. Secondly, the *inner function* $\psi_{p,q}$ is chosen independent from f but is highly non-smooth even though it is continuous. Moreover, the original proof of Theorem 4.1 is non-constructive giving little information about the inner and outer functions. In 1991, Kůrková made the connection more precisely by giving a proof of Theorem 3.6 using the Superposition theorem as a deep neural network. Furthermore, improved versions of Theorem 4.1 in [20], [30], [31], [4] and [5] paved the way for a stronger connection between neural networks and the Superposition theorem [24], [13], [18]. To further strengthen the relation between KA representation and neural networks, we need to construct a new KA representation which is equivalent to a deep neural network.

We now summarize the discussion so far. To estimate a d-variate β-smooth regression function f with positive number $\beta \leq 1$ up to error $m^{-\beta}$, we need an order of m^d parameters to parametrize f. We want to construct a deep neural network which uses at most an order of m^d parameters to approximate f up to an error $m^{-\beta}$. Moreover, this deep neural network mimics the KA representation. i.e., our construction is a rewriting of a d-variate function into a composition of univariate functions.

5 An extension of the KA representation

To construct a deep neural network as described above, we first need a KA representation which we can imitate. We get a new KA representation from [28], Section 2. In this section we extend the domain $[0,1]^d$ on which the new KA representation is proven and check if we obtain an equal order of approximation rates. Since we only extend the domain, the proofs of Theorem 5.3, Theorem 5.5 and Theorem 5.6 are similar as the original proof in [28]. We first consider the domain $[a,b]^d \subset \mathbb{R}^d$ with $a < b$. Our goal is to find an approximation of β-Hölder-smooth functions $f : [a,b]^d \to \mathbb{R}$. The new KA representation uses *space-filling curves* which are defined as follows.

Definition 5.1 (Space-filling curve)**.** *Let $d \geq 2$ be an integer. For non-empty compact domain $D \subset \mathbb{R}^d$, a space-filling curve is a surjective map $\gamma : [0,1] \to D$.*

Popular examples of space-filling curves on the unit cube are the Hilbert curve and the Peano curve ([11], [22]). If the inverse γ^{-1} exists, we can rewrite a function $k : [0,1]^d \to \mathbb{R}$ as follows:

$$k = (k \circ \gamma) \circ \gamma^{-1}.$$

For f, we map the domain $[a,b]$ to $[0,1]$ by transformation $T(x) := (x-a)/(b-a)$. We can extend this transformation so that it holds for d dimensions. We have $T_d : [a,b]^d \to [0,1]^d$ defined by $T_d(\mathbf{x}) := (T(x_1), \ldots, T(x_d))$. We rewrite

$$f = (f \circ T_d^{-1} \circ \gamma) \circ (\gamma^{-1} \circ T_d) = g \circ \Gamma, \text{ with } g := f \circ T_d^{-1} \circ \gamma, \text{ and } \Gamma := \gamma^{-1} \circ T_d.$$

This gives an outer function g that depends on f and an inner function $\Gamma : [a,b]^d \to [0,1]$. This inner function is d-variate so it differs from the original KA representation Theorem 4.1. A diagram below shows what happens.

$$\begin{array}{ccc} [a,b]^d & \xrightarrow{\quad f \quad} & \mathbb{R} \\ {\scriptstyle T_d}\downarrow & \searrow^{\Gamma} & \uparrow{\scriptstyle g} \\ [0,1]^d & \xrightarrow[\quad \gamma^{-1} \quad]{} & [0,1] \end{array}$$

The main difficulty of space-filling curves with respect to the KA representation is Netto's theorem ([17] Proposition 4.4).

Theorem 5.2 (Netto's theorem)**.** *A continuous surjective map $\gamma : [0,1] \to [0,1]^2$ cannot be injective.*

Theorem 5.2 seems to make it impossible to find a continuous inner function, i.e. we cannot find a continuous map Γ^{-1}. In total we now have two problems to rewrite f in a KA representation using space-filling curves. First, the inner function is a d-variate function. Secondly, the inner function is discontinuous by Netto's theorem. The following theorem overcomes the first problem. It is an extension of Lemma 1 in [28]. The proof uses the B-adic representation for choices of $B \in \mathbb{Z}_{\geq 2}$. e.g. for $B = 2$ we have binary number representation and for $B = 10$ we have the decimal number system.

Theorem 5.3. *Fix integers $d, B \in \mathbb{Z}_{\geq 2}$ and $a, b \in \mathbb{R}$ with $a < b$. There exists a monotone function $\psi : [a, b] \to \mathbb{R}$ such that for any function $f : [a, b]^d \to \mathbb{R}$ we can find a function $g : \mathbb{R} \to \mathbb{R}$ that satisfies*

$$f(x_1, \ldots, x_d) = g\left(\sum_{p=1}^{d} B^{-p}\psi(x_p)\right).$$

Proof. First, rewrite each $x \in [a, b]$ in its B-adic representation. This is however not unique e.g. for $B = 10$ we have $1 = 0.\overline{9}$. Therefore, we select one B-adic representation for each x. For $n_j^x \in \{0, \ldots, B-1\}$ we have the following B-adic representation,

$$x = \sum_{j=1}^{\infty} \frac{n_j^x}{B^j} =: [0.n_1^x n_2^x n_3^x \ldots]_B.$$

Secondly, we map x to $y = T(x) = (x - a)/(b - a) \in [0, 1]$. In B-adic representation we have,

$$y = T(x) = T\left(\sum_{j=1}^{\infty} \frac{n_j^x}{B^j}\right) = T\left([0.n_1^x n_2^x n_3^x \ldots]_B\right) = [0.m_1^y m_2^y m_3^y \ldots]_B.$$

This gives a unique B-adic representation for each $T(x)$ in $[0, 1]$ and $m_j^y \in \{0, \ldots, B-1\}$. Thirdly, we define $\tau : [0, 1]$ by

$$\tau(z) := \sum_{j=1}^{\infty} \frac{n_j^z}{B^{d(j-1)}} = [n_1^z.0 \ldots 0 n_2^z 0 \ldots 0 n_2^z 0 \ldots]_B \text{ for } z \in [0, 1].$$

Where in B-adic notation we have $(d-1)$-many zeros between each n_j^z, $j = 1, 2, \ldots$. The function τ is monotone increasing since we fixed $d, B \geq 2$ and transformation T is monotone increasing as $a < b$. The composition $\psi := \tau \circ T$ is therefore monotone increasing as well. Lastly, we define $\Psi : [a, b]^d \to [0, 1]$ by

$$\Psi(x_1, \ldots, x_d) := \sum_{p=1}^{d} B^{-p}\psi(x_p) = [0.m_1^{y_1} m_1^{y_2} \ldots m_1^{y_d} m_2^{y_1} \ldots]_B.$$

Reversing the steps recovers $x_1, \ldots, x_d$ from Ψ. So the inverse Ψ^{-1} exists. Defining $g := f \circ \Psi^{-1}$ concludes the statement. □

We now have that $\Psi^{-1} : [0, 1] \to [a, b]^d$ is a space-filling curve. Furthermore, the inner function Ψ consists of additive univariate *interior functions* ψ hence the inner function itself is univariate. We therefore overcome our first problem as discussed above.

The inner function is however still discontinuous for finite B-adic representation. Set for example $x = 1/2$. We have for limit $x \downarrow 1/2$,

$$\Psi^{-1}(x) \to (T^{-1}(1/2), T^{-1}(0), \ldots, T^{-1}(0)) = (b/2 - 3a/2, -a, \ldots, -a)$$

and for limit $x \uparrow 1/2$,

$$\Psi^{-1}(x) \to (T^{-1}(1/2), T^{-1}(1), \ldots, T^{-1}(1)) = (b/2 - 3a/2, b - 2a, \ldots, b - 2a).$$

Which is only equal if and only if $a = b$. This makes ψ also discontinuous. Function Ψ seems unclear at first glance but it works similar as Hilbert's hotel [16] where for input $T_d(x_1, \ldots, x_d)$ in B-adic representation we assign $m_1^{y_1}$ to the first hotel room, $m_1^{y_2}$ to the second and $m_j^{y_i}$ to the $j(i+1)$'th room. For $B = d = 2$ we get the Morton order coinciding up to rotation with the Z-order curve ([21], [2]). To overcome the second problem of discontinuity, we define g on the domain of the *Cantor set*.

Definition 5.4 (Cantor set)**.** *The Cantor set $\mathcal{C} \subset [0,1]$ is defined as*

$$\mathcal{C} := [0,1] \setminus \bigcup_{n=1}^{\infty} \bigcup_{k=0}^{3^{n-1}-1} \left(\frac{3k+1}{3^n}, \frac{3k+2}{3^n} \right).$$

It is the set for which iteratively, the open middle third is removed of a line segment in $[0,1]$, infinitely many times. Using the B-adic representation above, we can rewrite $\mathcal{C}$ as

$$\mathcal{C} = \{x \in [0,1] : [0.n_1^x n_2^x n_2^x \ldots]_3 \text{ with } n_i^x = \{0,2\}, \text{ for all } i = 1,2,\ldots\}.$$

This gives the following extension of Theorem 5.3.

Theorem 5.5. *For fixed dimension $d \geq 2$, $a, b \in \mathbb{R}$ with $a < b$, there exists a monotone function $\phi : [0,1] \to \mathcal{C}$ such that for any function $f : [a,b]^d \to \mathbb{R}$ we can find a function $g : \mathcal{C} \to \mathbb{R}$ such that*

(1) $f(x_1, \ldots, x_d) = g\left(3 \sum_{p=1}^{d} 3^{-p} \phi(T(x_p))\right)$;

(2) if f is continuous, then g as well;

(3) if there exists positive $\beta \leq 1$ and a constant Q, such that $|f(\mathbf{x}) - f(\mathbf{y})| \leq Q\|\mathbf{x} - \mathbf{y}\|_\infty^\beta$ for all $\mathbf{x}, \mathbf{y} \in [a,b]^d$, then, $|g(x) - g(y)| \leq (2(b-a))^\beta Q |x - y|^{\frac{\beta \log 2}{d \log 3}}$ for all $x, y \in \mathcal{C}$.

Proof. We prove *(1)* by using the same construction as in Theorem 5.3. We now associate each $y = T(x) \in [0,1]$ with a 2-adic representation and define

$$\phi(y) := \sum_{j=1}^{\infty} \frac{2m_j^y}{3^{1+d(j-1)}} = [0.(2m_1^y) \overbrace{0 \ldots\ldots\ldots 0}^{(d-1)-\text{times}} (2m_2^y) 0 \ldots\ldots\ldots]_3,$$

$$\Phi(y_1, \ldots, y_d) := 3 \sum_{p=1}^{d} 3^{-p} \phi(y_p) = [0.(2m_1^{y_1})(2m_1^{y_2}) \ldots (2m_2^{y_1})(2m_2^{y_2}) \ldots]_3.$$

For all $j = 1, 2, \ldots$ we have the 3-adic representation of ϕ with $m_j^y \in \{0, 2 \cdot 1\}$. By definition of the Cantor set we can conclude that the image of ϕ is in $\mathcal{C}$. Moreover, the function $\Phi : [a,b]^d \to \mathcal{C}$ is

also mapped correctly to the Cantor set. Using the same construction as Theorem 5.3 we deduce that ϕ is monotone increasing and we can reverse the construction so that Φ^{-1} exists. Finally, we define $g := f \circ \Phi^{-1} : \mathcal{C} \to \mathbb{R}$ and conclude *(1)*.

Next, to prove *(2)* and *(3)* we show that

$$\|\Phi^{-1}(x) - \Phi^{-1}(y)\|_\infty \leq 2(b-a)|x-y|^{\log 2/d \log 3}, \text{ for all } x, y \in \mathcal{C}.$$

First, the space-filling curve Φ^{-1} maps each point $[0.y_1y_2\ldots]_2 \in \mathcal{C}$ to the d-dimensional vector

$$\begin{aligned}(T^{-1}([0.(y_1/2)(y_{d+1}/2)\ldots]_2), \ldots, T^{-1}([0.(y_d/2)(y_{2d}/2)\ldots]_2))^\top \\ = (b-a)([0.x_1x_{d+1}\ldots]_2, \ldots, [0.x_d, x_{2d}, \ldots]_2)^\top + a \in [a,b]^d.\end{aligned}$$

With $T^{-1}(y_i) = (b-a)x_i + a$ for all $i = 1, \ldots, d$. Hence the inverse expands the space in $\mathbb{R}^d$ by a factor of $(b-a)$. Secondly, suppose $x, y \in \mathcal{C}$ and define positive integer $k^*(x,y)$ for which

$$3^{-(k+1)d} \leq |x-y| < 3^{-kd}.$$

It denotes the number for which the first $d \cdot k^*$ ternary digits of x and y coincide. For $\Phi^{-1}(x)$ and $\Phi^{-1}(y)$ it denotes the first k^* binary digit that agree in each component. Therefore, by expansion of T^{-1} and the definition of k^* we have

$$\begin{aligned}\left\|\Phi^{-1}(x) - \Phi^{-1}(y)\right\|_\infty &\leq \Big\| \big[(b-a)([0.x_1x_{d+1}\ldots]_2, \ldots, [0.x_d, x_{2d}, \ldots]_2)^\top + a \\ &\qquad -(b-a)([0.y_1y_{d+1}\ldots]_2, \ldots, [0.y_d, y_{2d}, \ldots]_2)^\top - a\big] \Big\|_\infty, \\ &= (b-a)\Big\| \big[([0.x_1x_{d+1}\ldots]_2, \ldots, [0.x_d, x_{2d}, \ldots]_2)^\top \\ &\qquad -([0.y_1y_{d+1}\ldots]_2, \ldots, [0.y_d, y_{2d}, \ldots]_2)^\top\big] \Big\|_\infty, \\ &\leq (b-a)2^{-k^*} = 2(b-a)2^{-(k^*+1)} = 2(b-a)\left(3^{-(k^*+1)d}\right)^{\log 2/d\log 3}, \\ &\leq 2(b-a)|x-y|^{\log 2/d \log 3}.\end{aligned}$$

The map Φ^{-1} is thus $(\log 2)/(d \log 3)$-Hölder-smooth with constant $2(b-a)$ so it is continuous. If f is continuous, then the composition g is continuous as well and we conclude statement *(2)*. Lastly, if f is β-smooth for some positive $\beta \leq 1$, then for all $x, y \in \mathcal{C}$ we have,

$$|g(x) - g(y)| = \left|f(\Phi^{-1}(x)) - f(\Phi^{-1}(y))\right| \leq Q\left(\|\Phi^{-1}(x) - \Phi^{-1}(y)\|_\infty\right)^\beta \leq (2(b-a))^\beta Q|x-y|^{\frac{\beta \log 2}{d \log 3}}.$$

This concludes the last statement. □

Restricting the space-filling curve to the Cantor set makes it continuous and we avoid the conclusion of Netto's theorem. We now have overcome the two problems discussed above and the KA representation is extended to the $[a,b]^d$-domain. This KA representation has continuous outer function g and additive continuous inner function Φ. Specifying on the space-filling curve of this KA representation, our space-filling curve Φ^{-1} is the Z-order curve. Next, we look if this new representation gives a good approximation for f. Even though Φ is continuous, the interior function ϕ is discontinuous for the same reason as ψ from Theorem 5.3 is discontinuous. To get an approximation for f, we truncate ϕ. This also reduces the complexity of the discontinuous interior function. We obtain the following approximation.

Theorem 5.6. *Let $d \geq 2$, $a, b \in \mathbb{R}$ with $a < b$ and suppose positive integer K. For a binary representation $x \in [a,b]^d$ and $y = T(x)$ define*

$$\phi_K(y) := \sum_{j=1}^{K} 2m_j^y 3^{-1-d(j-1)}.$$

If there exists $\beta \in (0,1]$ and a constant Q such that $|f(\mathbf{x}) - f(\mathbf{y})| \leq Q\|\mathbf{x}-\mathbf{y}\|_\infty^\beta$ for all $\mathbf{x}, \mathbf{y} \in [a,b]^d$, then, we can find univariate function g such that

(1) $|g(x) - g(y)| \leq (2(b-a))^\beta Q|x-y|^{(\beta \log 2)/(d \log 3)}$ for all $x, y \in \mathcal{C}$;

(2) for all $\mathbf{x} = (x_1, \ldots, x_d)^\top \in [a,b]^d$,

$$\left| f(\mathbf{x}) - g\left(3\sum_{p=1}^{d} 3^{-p}\phi_K(T(x_k))\right)\right| \leq 2(b-a)Q2^{-\beta K}.$$

Proof. We let ϕ and g be as in Theorem 5.5(1) and ϕ_K defined as above. This returns a sufficient g for *(1)*. For *(2)*. Since ϕ is in the Cantor set, the truncated ϕ_K is in $\mathcal{C}$ as well and have the same first Kd ternary representation. We have

$$\begin{aligned}\left| f(\mathbf{x}) - g\left(3\sum_{p=1}^{d} 3^{-p}\phi_K(T(x_p))\right)\right| &= \left| g\left(3\sum_{p=1}^{d} 3^{-p}\phi(T(x_p))\right) - g\left(3\sum_{p=1}^{d} 3^{-p}\phi_K(T(x_p))\right)\right|, \\ &\leq \quad (2(b-a))^\beta Q \left|3\sum_{p=1}^{d} 3^{-p}\left(\phi(T(x_p)) - \phi_K(T(x_p))\right)\right|^{\frac{\beta \log 2}{d \log 3}} = \star.\end{aligned}$$

Because ϕ and ϕ_k have the same first Kd ternary digits we obtain

$$\sum_{p=1}^{d} \phi(T(x_p)) - \phi_K(T(x_p)) = \sum_{p=1}^{d}\sum_{j=Kd+1}^{\infty} \frac{2m_j^{y_p}}{3^{1+d(j-1)}} \leq 2\sum_{q=Kd+1}^{\infty} 3^{-q}.$$

Continuing, from the geometric sum formula $\sum_{q=0}^{\infty} 3^{-q} = 3/2$, we can conclude statement *(2)* by,

$$\begin{aligned}\star &\leq (2(b-a))^\beta Q \left|2\sum_{q=Kd+1}^{\infty} 3^{-q}\right|^{\frac{\beta \log 2}{d \log 3}}, \\ &\leq (2(b-a))^\beta Q \left|2 \cdot 3^{-Kd+1}\sum_{q=0}^{\infty} 3^{-q}\right|^{\frac{\beta \log 2}{d \log 3}} = (2(b-a))^\beta \left|3^{-Kd}\right|^{\frac{\beta \log 2}{d \log 3}}, \\ &\leq \ 2(b-a)Q2^{-\beta K}.\end{aligned}$$

Truncating ϕ up to K digits gives an approximation for f. Comparing Theorem 5.6 to Lemma 4 in [28], we get equal approximation up to a factor $(b-a)$. Therefore, wee can conclude that the domain is of influence to the approximation rate. The interior function ϕ_K is by truncation still discontinuous and the outer function g is still f depended. In Section 6 we overcome these last two problems by replacing ϕ_K with a deep ReLU network and show that the outer function can be well-approximated by a shallow ReLU network.

5.1 Other compact domains for the KA representation

We have now extended the KA representation from [28] using space-filling curves on the general d-dimensional cube. The representation of Theorem 5.5 is up to transformation T_d equivalent to the KA representation of [28]. In each component dimension we were able to do the same smooth transformation from $[a, b]$ to $[0, 1]$. Thus the $[a, b]^d$-domain is not of major influence to the KA representation. What happens to the KA representation if we consider other compact domains in $\mathbb{R}^d$? Will the KA representation differ from Theorem 4.1 and Theorem 5.5? We now look at the d-dimensional closed ball which is defined as

$$\mathbb{B}^d = \{x \in \mathbb{R}^d : \|x\|_2 \leq 1\},$$

and try to obtain a similar KA representation for β-smooth functions f on $[0, 1]^d$. To do this we use the same strategy as for the $[a, b]^d$-domain. Topologically, these spaces are homeomorphic to each other, thus there exists a homeomorphism between the unit cube and closed ball. A first choice would be a homeomorphism of the following theorem.

Theorem 5.7. *Let $d \geq 1$ be an integer. The function $h_1 : \mathbb{B}^d \to [-1, 1]^d$ given by*

$$h_1(\mathbf{x}) = \frac{\|\mathbf{x}\|_2 \mathbf{x}}{\max\{|x_1|, \ldots, |x_d|\}} \mathbb{1}\{\mathbf{x} \neq \mathbf{0}\}, \text{ is homeomorphic.}$$

Proof. First we show bijectivity. $h_1(\mathbf{x}) = \mathbf{0}$ if and only if $\mathbf{x} = \mathbf{0}$ by definition of h_1. Secondly, we note that $\|\mathbf{x}\|_2 / \max\{|x_1|, \ldots, |x_d|\} \in \mathbb{R}$. Therefore, we have $h_1(\mathbf{x}) = \lambda \mathbf{x}$ for $\lambda \in \mathbb{R}$. A line segment from the origin to the boundary of the ball is extended by λ as a line segment from the origin to the boundary of the d-cube (see Figure 5). The set of all the line segments from the origin to the boundary of the ball contain all the points in $\mathbb{B}^d$. Furthermore, each line in this set only intersects each other at $\mathbf{0}$. Hence for each point $\mathbf{p} \in \mathbb{B}^d \setminus \{\mathbf{0}\}$ we can find a unique line segment that passes through $\mathbf{p}$. This gives a unique mapping of h_1 to the unit cube as $h_1(\mathbf{p}) = \lambda \mathbf{p}$, the unique point in the line segment from the origin to the unit cube that passes through $h_1(\mathbf{p})$. The map h_1 is by these two points bijective. Next, the inverse is given by $h_1^{-1}(\mathbf{x}) = \lambda^{-1}\mathbf{x}$ through the previous arguments. If we take a point $h_1(\mathbf{p}) \subset [-1, 1]^d$ with open neighbourhood U of $h_1(\mathbf{p})$ contained in $[-1, 1]^d$, then U consists of multiple open intervals of line segments. Furthermore, the inverse $h_1^{-1}U = \lambda^{-1}U$ is open on the ball, hence it is continuous. We conclude the statement. □

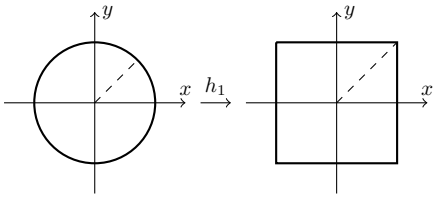

Figure 5: Example for $d = 2$. A line segment from the origin to the boundary in $\mathbb{B}^2$ is extended to the boundary of $[-1, 1]^2$ by h_1.

The problem with h_1 is that the max function can not be decomposed as a summation of compositions of one-variable functions. If we try to use h_1 in a similar fashion as T_d from the previous subsection we lose the benefit of Theorem 5.3. The inner function cannot be rewritten as a summation of univariate functions and the inner function remains d-variate. To find better options, we consider $d = 2$ and look at the map

$$h_2(x, y) = (x, x^2 + y^2).$$

This map can be decomposed additively for $e(x) = x^2$ with $h_2(x, y) = (x, e(x) + e(y))$. However, it is not bijective because $h_2(x, y) = h_2(x, -y)$ (see Figure 6). The following map from [8] is bijective.

Theorem 5.8. *Let $d = 2$. The map $h_3 : \mathbb{B}^2 \to [-1, 1]^2$ given by*

$$h_3(x, y) = \left(x, \frac{y}{\sqrt{1 - x^2}}\right),$$

is bijective.

Proof. First, we note that the image of $y = \sqrt{1 - x^2}$ for $x \in [-1, 1]$ is the half-circle for $x \geq 0$ with radius 1 from the origin. Therefore, $y/\sqrt{1 - x^2}$ is a ratio which depends on the radius for the point $(x, y) \in \mathbb{B}^2$. Using $c^2 = x^2 + y^2$ for radius $c \in [0, 1]$, we can rewrite the map h_3 as follows:

$$h_3(x, y) = \left(x, \frac{y}{\sqrt{1 - x^2}}\right) = \left(x, \sqrt{\frac{c^2 - x^2}{1 - x^2}}\right).$$

Secondly, suppose $h_3(x_1, y_1) = h_3(x_2, y_2)$ for $(x_1, y_1), (x_2, y_2) \in \mathbb{B}^2$. This means $x_1 = x_2$ and therefore, $y_1 = y_2$ using the rewriting of h_3 above. This proves injectivity. For surjectivity, suppose $(x_2, y_2) \in [-1, 1]^2$. We set $x_1 = x_2$ and $y_1 = y_2\sqrt{1 - x_2^2}$. This gives

$$h_3(x_1, y_1) = h_3\left(x_2, y_2\sqrt{1 - x_2^2}\right) = \left(x_2, \frac{y_2\sqrt{1 - x_2^2}}{\sqrt{1 - x_2^2}}\right) = (x_2, y_2).$$

It is left to show that (x_1, y_1) is indeed a point in $\mathbb{B}^2$. We have

$$c^2 = x_1^2 + y_1^2 = x_2^2 + \left(y_2\sqrt{1 - x_2^2}\right)^2 = x_2^2 + y_2^2 - x_2^2 y_2^2 \leq 1.$$

Where we use the inequality of $(x_2, y_2) \in [-1, 1]^2$. This proves surjectivity and we conclude the statement. □

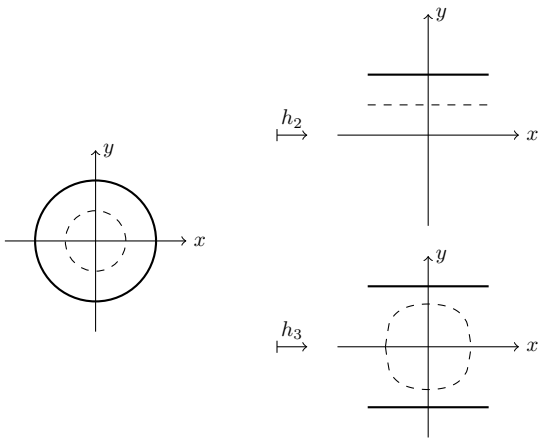

Figure 6: Visualisation of functions h_2 and h_3 from the ball to the unit cube. For the dashed contour, the radius is 0.5.

Again as with h_1, the function h_3 is not a composition of additive functions so it is difficult to incorporate h_3 in a KA representation. For h_2 we can also look at the positive side of the ball $\mathbb{B}^d_+ := \mathbb{B}^d \cdot \mathbb{1}\{x_d \geq 0\}$. This will give us the desired homeomorphism: we define $h_4 : \mathbb{B}^d_+ \to [0,1]^d$ by

$$h_4(x_1, \ldots, x_d) := \left(\frac{x_1 + 1}{2}, \frac{x_2 + 1}{2}, \ldots, \frac{x_{d-1} + 1}{2}, \sqrt{x_1^2 + \ldots + x_d^2} \right).$$

This function is bijective as all its individual components are now bijective on each individual component domain. Setting $r(x) = (x+1)/2$, $s(x) = \sqrt{x}$ and $t(x) = x^2$, we can rewrite h_4 as

$$h_4(x_1, \ldots, x_d) = \left(r(x_1), \ldots, r(x_{d-1}), s\left(\sum_{p=1}^{d} t(x_p) \right) \right).$$

Function h_4 now transforms each component of $\mathbb{B}^d_+$ to each component in $[0,1]^d$. With respect to the map T_d from the previous subsection we have a similar transformation up to the last component. Hence if we now find a new KA representation on the domain $\mathbb{B}^d_+$, it will differ from the KA representation we have seen so far. This KA representation is more complex as our transformation in the last component is more complex (i.e. we have more compositions of univariate functions).

6 Applying the extended KA representation

In the previous section we obtained an approximation of a β-smooth function f from the cube $[a,b]^d$ to $\mathbb{R}$:

$$f(\mathbf{x}) \approx g\left(3\sum_{p=1}^{d} 3^{-p}\phi_K(T(x_p))\right), \text{ with } \phi_K(x) = \sum_{j=1}^{K} 2a_j^x 3^{-1-d(j-1)}.$$

The interior function ϕ_K is still discontinuous and the outer function is f depended. Setting $a = 0, b = 1$, then T is the identity function. We can approximate ϕ_K with a deep ReLU network from [28], Section 3. Using this deep ReLU network we apply gradient descent to the above approximation and investigate if we get the expected approximation rate using a suitable amount of network parameters.

6.1 Gradient descent

In Section 2 we introduced the non-parametric model. We have data $(\mathbf{X}_i, Y_i)_{i=1,\dots,n}$ and our goal is to find the best fitting regression function f such that for all $i = 1, \dots, n$, we have

$$Y_i = f(\mathbf{X}_i, \theta) + \varepsilon_i \text{ with } \varepsilon_i \sim \mathcal{N}(0,1),$$

where θ is the real parameter vector. This regression model is non-parametric as we assumed our regression function is in the parameter space which consists of β-Hölder-smooth functions. Next, in Section 3 we defined deep ReLU networks $\hat{f}$ to approximate the regression function. We use the least squared loss as our measure of fit. For V a real vector space, we want to find parameter vector

$$\hat{\theta}_n \in \text{argmin}_{\theta \in V} \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{f}(\mathbf{X}_i, \theta)\right)^2.$$

In general, we can rewrite the above functional as the loss $L(\theta; \mathbf{X}_i, Y_i)$. To obtain small loss, stochastic gradient descent (SGD) is the most popular choice. In short, SGD iteratively updates the weights and bias parameters of a neural network by moving them in the direction of the negative gradient of the loss. For this, the ReLU activation has to be differentiable which it is on $\mathbb{R}\setminus\{0\}$. For the point on the origin, we can fix 0 or 1 as the differential at this point. We update the parameter weights using

$$\theta_i := \theta_{i-1} - \alpha \nabla L(\theta; \mathbf{X}_i, Y_i).$$

Updating the parameters of $\hat{f}$ using SGD leads to a local minimum for the loss. We get a local minimum as deep neural networks are in general non-convex. SGD can be computed for neural networks using the backpropagation algorithm [26]. For the scope of this thesis we leave out these details.

In SGD, α is called the *step size* or *learning rate*. It determines the change of weights from the gradient. Choosing the correct learning rate is difficult and depends on the neural network. A learning rate which is too small gives little training whereas a high learning rate results in oscillations and in turn gives a non optimal minimum. We choose Adam [14] as our SGD optimizer. It uses an adaptive learning rate for each network parameter. The Adam optimizer is developed for large data sets and high-dimensional parameter space.

In total, for input $\mathbf{X}_i \in [0,1]^d$ and output $Y_i \in \mathbb{R}$ we wish to approximate a β-smooth function $f : [0,1]^d \to \mathbb{R}$ using the KA approximation of Theorem 5.5 with $T(x) = x$. We use the Adam optimizer with least squares loss and examine that our estimator contains m^d parameters to approximate f with expected rate of $m^{-\beta}$.

6.2 Constructing a deep ReLU network based on the KA representation

After discussing SGD, we now construct a deep ReLU network by following the construction in [28], Section 3. We first build a deep ReLU network to approximate the interior function ϕ_K. Concatenating all interior functions gives a network for the inner function. Afterwards, we extend the network so that it approximates the outer function g. A network here can also be referred to as a model.

6.2.1 Constructing a deep ReLU network for the interior function

To compute ϕ_K in a deep ReLU network we want to extract each bit a_j^x from the binary input $x = [0.a_1^x a_2^x \ldots]_2$ for $j = 1, \ldots, K$. We do this by using threshold activation function $\mathbb{1}\{x \geq 1/2\} = \sigma(x)$ and the identity activation function $\mathrm{id}(x) = x$. Starting, we obtain the first bit by

$$\sigma([0.a_1^x a_2^x \ldots]_2) = \mathbb{1}\{[0.a_1^x a_2^x \ldots]_2 \geq 1/2\} = a_1^x,$$

and then compute

$$2\mathrm{id}(x) - 2\sigma(x) = 2(x - a_1^x) = [0.a_2^x a_3^x \ldots]_2.$$

Iterating through these steps gives us a_j^x for $j = 1, \ldots, K$. To convert it to a ReLU network we note that the input is non-negative so we can change the identity activation function by the ReLU activation function. Additionally, we can change the threshold activation function by two ReLU activation functions for choice of $\varepsilon \downarrow 0$ with

$$\mathbb{1}\{x \geq 1/2\} = \frac{1}{\varepsilon} \max\left\{x - \frac{1-\varepsilon}{2}, 0\right\} - \frac{1}{\varepsilon} \max\left\{x - \frac{1+\varepsilon}{2}, 0\right\} =: \sigma_1(x) - \sigma_2(x).$$

This gives us three individual ReLU networks in each hidden layer (see Figure 7). Two networks are used to extract each bit from the input with architecture $(2, (2,1,1,1))$, $(2, (2,1,2,1))$ and one network to compute the result with architecture $(2, (2,1,1,1))$. As we want to extract K digits from the input, our interior function is a deep ReLU Network with $2K$ hidden layers and width 4 (see Figure 8 left). To sum up, from input $x = [0.a_1^x a_2^x \ldots]_2$ this deep ReLU network outputs $3\phi_K(x)$ with network architecture $(2K, (1,4,\ldots,4,1))$.

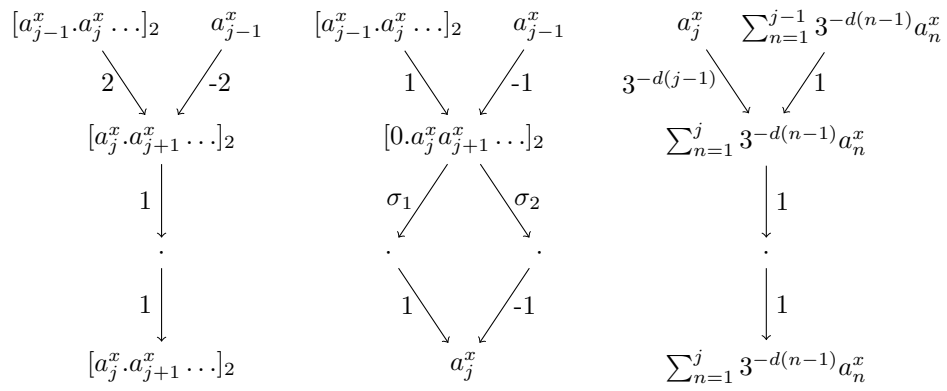

Figure 7: The individual models to compute $\phi_K(x)$

6.2.2 Constructing the full model

Having constructed the deep ReLU network to compute ϕ_K, we now build the full network to estimate β-smooth functions f. For this we use the following theorem.

Theorem 6.1. *Schmidt-Hieber 2020 (Theorem 3 in [28]) Let $p \in [1,\infty)$. If there exists $\beta \leq 1$ and a constant Q, such that $|f(\mathbf{x})-f(\mathbf{y})| \leq Q|\mathbf{x}-\mathbf{y}|^\beta_\infty$ for all $\mathbf{x},\mathbf{y} \in [0,1]^d$. Then, there exisits a deep ReLU network $\tilde{f}$ with $2K+3$ hidden layers, network architecture $(2K+3,(d,4d,\ldots,4d,d,1,2^{Kd}+1,1))$ and all network weights bounded in absolute value by $2\max\{Kd,\|f\|_\infty\}\cdot 2^{K\max\{d,p\beta\}}$, such that*

$$\|f-\tilde{f}\|_p \leq 2\,(Q+\|f\|_\infty)\cdot 2^{-\beta K}.$$

Hence after computing the inner function $\phi_K(x_i)$ for each $i=1,\ldots,d$, we can approximate the outer function by a shallow neural network with ReLU activation function and width $2^{Kd}+1$. This gives in total a deep ReLU network with architecture $(2K+3,(d,4d,\ldots,4d,d,1,2^{Kd}+1,1))$ (see Figure 8 right). The weight initializations of the last hidden layer are to be sampled from a given distribution and all other network parameters are initialized by its corresponding construction of the inner function. Theorem 6.1 leads to the rate $2^{-K\beta}$ using of the order 2^{Kd} parameters.

6.2.3 Program

The full model is programmed in Python and can be found on github [32] which contains extra details. For this we used the Keras functional API module in TensorFlow ([7], [1]). The functional API gives us a lot of flexibility when programming the network as we do not have a fully connected deep neural network. The total model to approximate f consists of several smaller models combined together as described above. For each output graph, a special function is made. In total we have $3\cdot 2^{Kd}+4+(24K+33)d$ parameter weights for $K>2$. This gives indeed an order of 2^{Kd} parameters. Additionally, it imitates the KA approximation as desired.

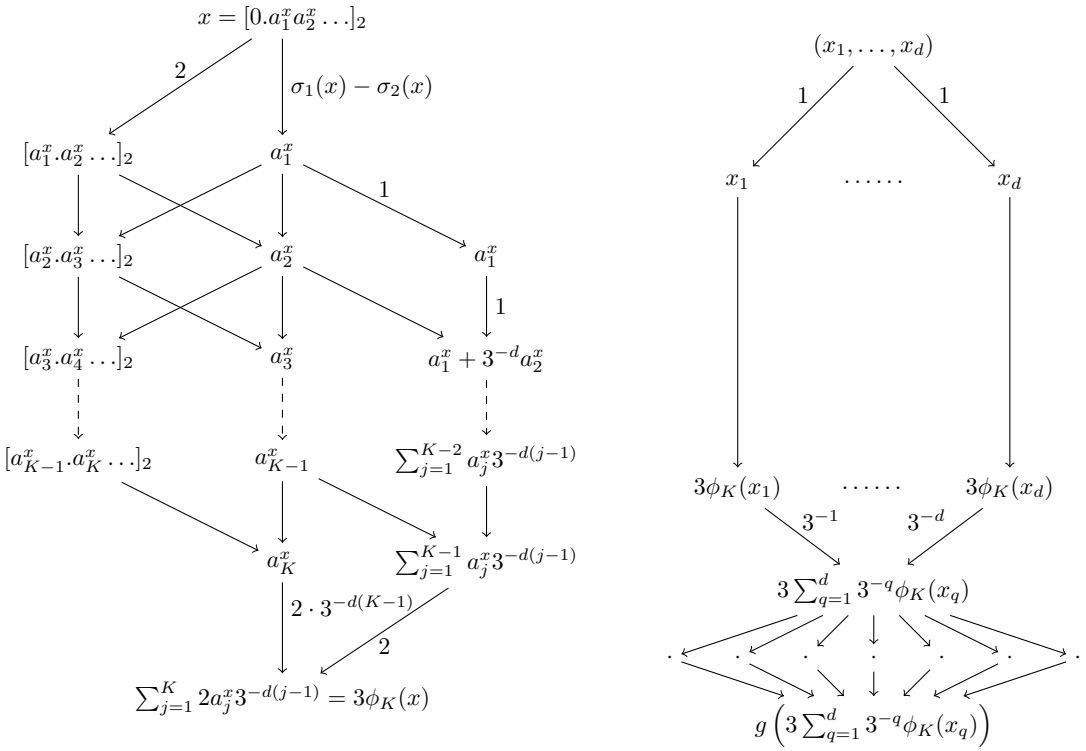

Figure 8: On the left side: the deep ReLU network to approximate ϕ_K using the three individual 2-hidden-layer models. On the right side: the total model from Theorem 6.1 to approximate β-smooth function f.

6.3 Results

Next, we will confirm that this deep ReLU network returns the expected approximation rate of $2^{-\beta K}$. Our data consists of 10,000 random points in $[0,1]^d$ which we split in 10% test and 90% train datasets. Moreover, we represent this dataset in binary form using the `binary_input` function. In any training we use 200 epochs. We do this because for most cases, after 200 epochs there is little to no training and in some cases it even increases the loss. As this is undesirable we ignore these cases. We view the result of training by plotting the model loss as a function of epochs. First we test the ϕ_K-network by making a prediction dataset from the random data including some noise. The noise takes samples of the normal distribution with variance 0.01. Adding noise gives the regression model from Definition 2.1. We train the model for $d = 1$, $K = 1, \ldots, 10$ and $\varepsilon = 10^{-1}, \ldots, 10^{-4}$.

Analyzing the results of the ϕ_K-model in Figure 9, we obtain no training for $K = 1$ and all ε. As $K = 1$, our output set consists of two points $\{0, 2 \cdot 1\}$. We truncated a binary input up to K digits so we get a maximum number in the output set of 2^K. With only two outputs, the model gets the best loss for choosing one of the two options. This results in no training.

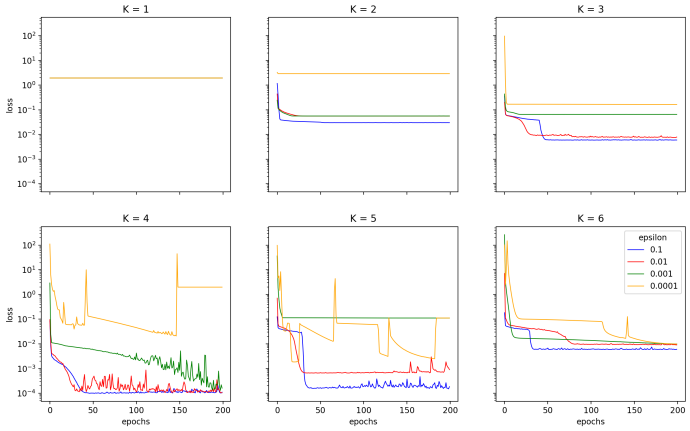

Figure 9: Loss versus epoch of ϕ_K with different choices of ε. All choices have same random input data set

For $K > 1$ we get different loss for all ε. As we decrease ε, the training results are more volatile. This result is justified by the model which computes each bit a_i^x for $i = 1, \ldots, K$ (Figure 7 middle). For any $\varepsilon \in (0, 1)$ we have $\sigma_1(x) - \sigma_2(x) = \mathbb{1}\{x \geq 1/2\}$. Therefore, the output of the truncated activation function is independent of ε. So why do we have different results? For σ_1, σ_2 we initialize the weights and biases as

$$\sigma_1(x) = \frac{1}{\varepsilon} \max\left\{x - \frac{1-\varepsilon}{2}, 0\right\} = \max\left\{\frac{1}{\varepsilon}x - \frac{1-\varepsilon}{2\varepsilon}, 0\right\} = \sigma_1\left(\frac{1}{\varepsilon}x - \frac{1-\varepsilon}{2\varepsilon}\right),$$

which for $\varepsilon \downarrow 0$ gives large initial weights and biases. Focusing on the learning rate, we have large shifts for the weights and biases even if the learning rate is low. Moreover, the learning rate is adaptive and so the weights and biases of σ_1 and σ_2 differ during training and in return may result in false digit output. Additionally, the output space is split up by the first digit. For large K, the output set is split between intervals $[0, 1)$ and $[2, 3)$. During training, the prediction may switch between these intervals and subsequently increases or decreases the loss significantly. For $K > 6$ we get similar results (Appendix A, Figure 13). As $\varepsilon = 0.1$ is the most stable option, we choose this ε in the next results.

One might argue that all options for ε gives insufficient loss and say we could get better results when we have different weight initialization for the interior function network. Hence we compare the ϕ_K-model with standard weight initialization. In Keras, the standard is the Glorot uniform initialization [6]. In this initialization, weights are set by drawing random samples from the uniform distribution on $\mathbb{R}$.

Examining the differences in loss between weight initialization in Figure 10, we note that for small K the Glorot uniform initializer performs better than our KA initialization. Doing multiple training (50 times) for the Glorot uniform model gives similar loss for $K = 1, 2, 3, 4$. Similar to $K = 1$. However, this happens infrequently. For $K > 4$ we get equivalent results for Glorot uniform (see Appendix Figure 15). Presumably, there is no training as the model is too deep and the number of random weights can not be initialized correctly.

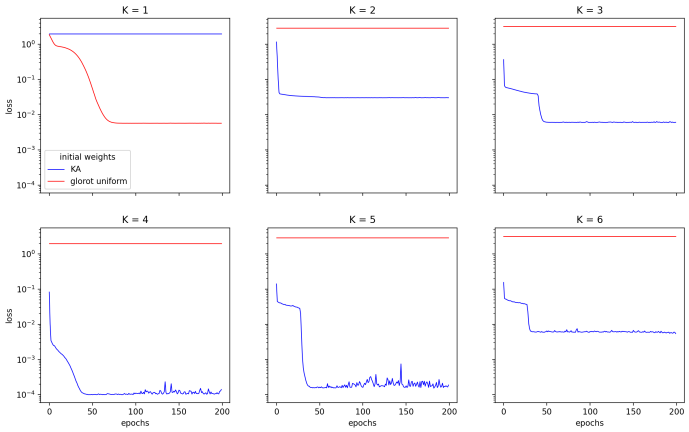

Figure 10: Loss versus epochs of ϕ_K with different weight initialization. All choices use the same input data set. For KA initialization, the weights are initialized with $\varepsilon = 0.1$.

The Glorot uniform initialization now scores best for picking one of the two output interval spaces. Continuing, we now check the loss for the total model. For this we look at three β-smooth functions on $[0,1]^d$:

$$f_1(\mathbf{x}) = \|\mathbf{x}\|_1, \quad f_2(\mathbf{x}) = \|\mathbf{x}\|_2 \quad \text{and} \quad f_\infty(\mathbf{x}) = \|\mathbf{x}\|_\infty.$$

As these functions are p-norms they are β-smooth by Theorem 2.3. When training the total model, we do not train the inner weights as they are initialized such that they give the expected outcome of the inner function. We sample the weight initialization of the last hidden layer from the normal distribution. The input data set is once more 10,000 random points in $[0,1]^d$ and the prediction dataset is $f(\text{input data}) + \text{noise}$. Both again are split in 10% test and 90% train datasets.

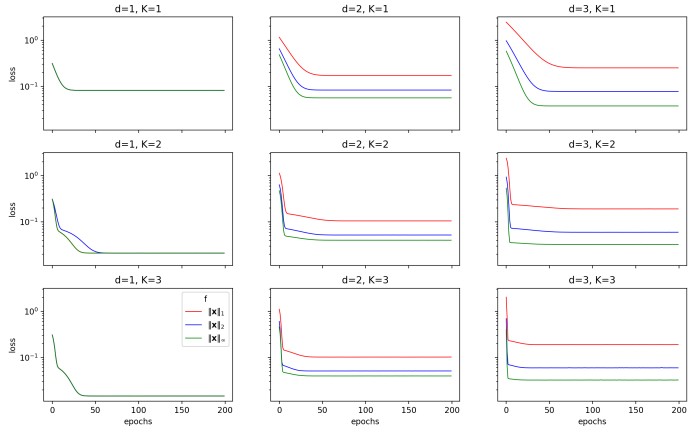

Figure 11: Loss versus epochs of the total model (Theorem 6.1) with $K = 1, 2, 3$ and $d = 1, 2, 3$. For each graph the same input dataset is used.

Figure 11 displays the loss of the total model. By Theorem 6.1, the approximation between f and the deep ReLU network $\hat{f}$ is bounded by $\|f\|_\infty$. For $d = 1$, the supremum of the three functions to approximate are equal so we get equal loss. Moreover, the 1-norm and ∞-norm are equal for $d = 1$ so there is no difference in their training. For $d \geq 2$ we remember that $\|\mathbf{x}\|_1 \geq \|\mathbf{x}\|_2 \geq \|\mathbf{x}\|_\infty$ for all $\mathbf{x} \in [0, 1]^d$. Therefore, the loss is also ordered in this sequence. In Figure 9 we saw that the inner function model can also be trained. Hence we do the following training: for the first 100 epochs we only train the parameters of the outer function, thereafter we train all model parameters for the last 100 epochs (Figure 12).

We only get significant extra training for $d = 1$ and $K > 1$. For $K = 1$ it is expected to get no extra training as this is in agreement of our results in Figure 9. As the input dimension increases, a local minimum has already been obtained by the outer function model. Hence the parameter training is done in the last hidden layer. We get similar results for $d > 3$ (Appendix Figure 14). By Theorem 6.1, we expected an approximation rate of $2^{-K\beta}$ when using 2^{Kd} parameters. Overall, we have up to $2^{3\cdot 3}$ parameters and get at least a loss of 2^{-3}. We even get a maximum loss of 2^{-6}.

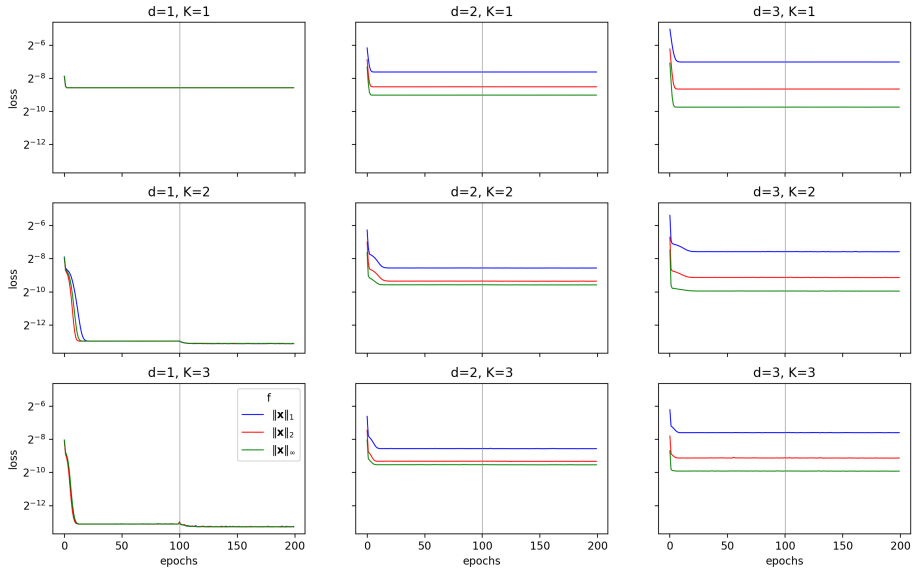

Figure 12: Loss versus epochs for the total model where after 100 epochs all weights and biases are trained.

7 Conclusion

Overall, this thesis is divided into three parts. The first part is introductory. We defined the non-parametric model to estimate a β-smooth regression function f for positive $\beta \leq 1$. The curse of dimensionality is stated as one of the main problems when finding an estimator for the non-parametric model. Next, we developed the concept of neural networks and discuss its connection to the curse of dimensionality. Thereafter, we introduced the Kolmogorov Arnold representation theorem and discussed the link with neural networks. Finally, we summarized the main objective of this thesis. Our goal is to construct a deep neural network mimicking the KA representation to approximate Hölder-smooth map f.

In the second part we strengthen the connection between the KA representation and neural networks. We do this by extending the KA representation of [28] to the compact domain $[a, b]^d$ using space-filling curves. Since we are able to smoothly transform the domain to $[0, 1]^d$, we were able to extend the KA representation. By truncating the interior function we get an approximation for Hölder-smooth functions f. The approximation differs up to a factor $(b-a)$ from the KA representation on $[0, 1]^d$. We tried to obtain a similar transformation from the d-dimensional ball but did not get the expected results. Concluding, we conjecture that the form of the KA representation depends on the domain.

In the last part we gave a construction of a deep ReLU network on the original $[0, 1]^d$-domain such that it indeed imitates the KA representation of Section 5. Training the model in Python we obtained four results. The first is the inconclusive choice of $\varepsilon \in (0, 1)$ for the interior function model. For now, it seems better to have a larger ε to have no oscillations during training. Secondly, compared to random weight initialization, the KA initialization is necessary for small loss. Thirdly, we received an expected optimal rate of $2^{-K\beta}$ with parameter order 2^{Kd} for small K and d. Lastly, the model training is done in the last hidden layer.

Even though the model has a good approximation rate with respect to the number of parameters, we can extend the training to obtain more insight on this KA representation. As a first extension, we need more training initialization just as in Appendix Figure 15. This gives a better overview of the minimum loss $\varepsilon \downarrow 0$. Moreover, we can confirm the true approximation rate of the KA representation. Secondly, the approximation rate has now been examined for small K and d. Will we get matching results when training for large K and d? We are however restricted to choice of K as real numbers are represented as floating points in Python. Thirdly, more β-Hölder-smooth functions are to be investigated. Specifically, functions which are Hölder-smooth for small β (e.g. $f(x) = x^\beta$). Alternatively, highly smooth functions are also of interest since the interior function is discontinuous. As a final extension, we wish to decrease the loss of the inner function model for any choice of K significantly. This is complicated due to the backpropagation algorithm. Weight updates in this algorithm are done in decimal form. As a consequence, the model loses its benefits when working with binary input: the weight initialization now is chosen such that we can compute the interior function using 2-adic numbers. Programming a custom backpropagation algorithm would be a good base for more research into this deep ReLU network.

Appendices

A Additional figure results

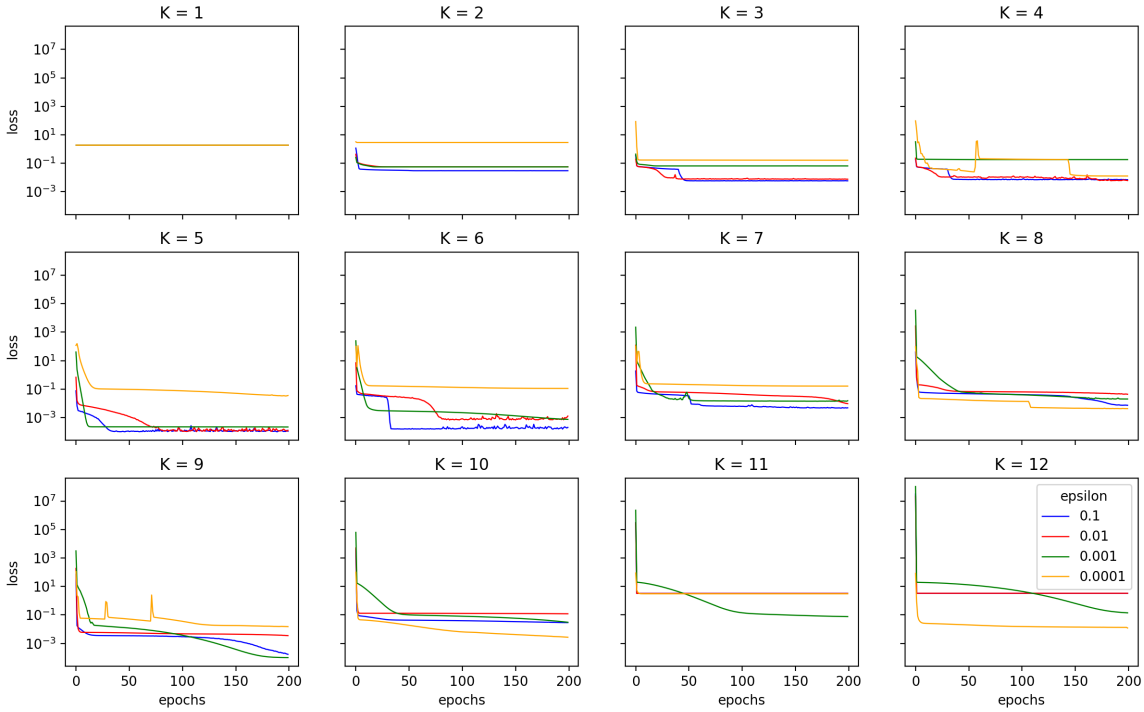

Figure 13: Loss versus epochs of ϕ_K-model with $d = 1$, $K = 1, \ldots, 12$ and $\varepsilon = 10^{-1}, \ldots, 10^{-4}$. For $K > 6$ we indeed still have indecisive results for choice of ε.

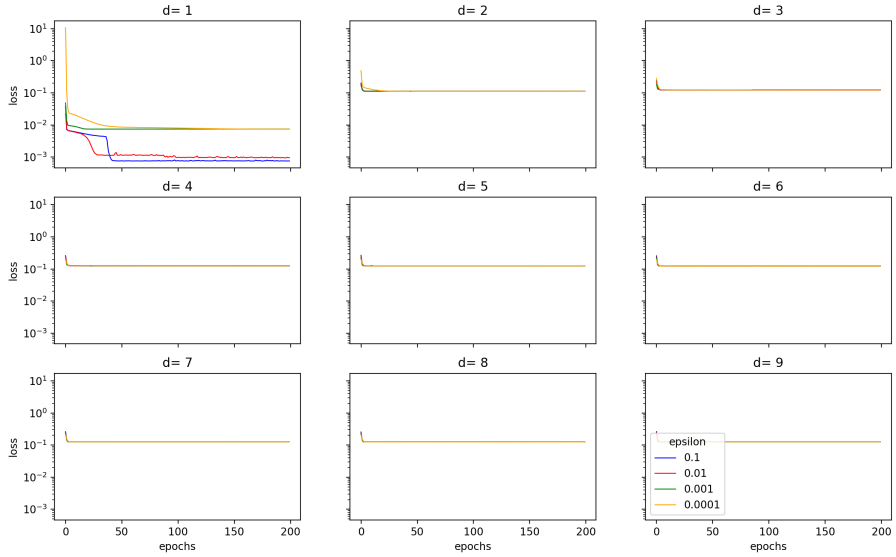

Figure 14: Loss versus epochs for the total model: $K = 3, \varepsilon = 0.1$ and $d = 1, \ldots, 12$. We see that for $d > 1$, the total model admits little to no additional training.

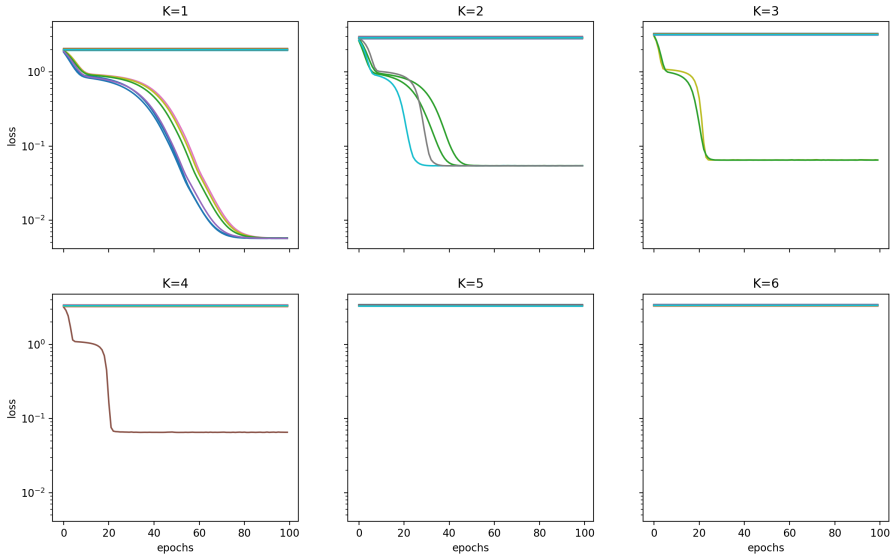

Figure 15: Loss versus epochs for the ϕ_K-model with Glorot uniform initialization.
For $K = 1, \ldots, 6$ we initialise the model 50 times. We see that for $K = 1$ we reach a minimum more frequently. As K gets larger, this happens less often. For comparison with the KA weight initialisation, as $K > 1$ we get a better loss.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[2] Michael Bader. *Space-filling curves*. Springer, Berlin, Heidelberg, 2013.

[3] Richard E Bellman. *Adaptive control processes*. Princeton university press, 2015.

[4] Jürgen Braun. *An Application of Kolmogorov's Superposition Theorem to Function Reconstruction in Higher Dimensions*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, dec 2009.

[5] Jürgen Braun and Michael Griebel. On a Constructive Proof of Kolmogorov's Superposition Theorem. *Constructive Approximation*, 30:653–675, 12 2009.

[6] François Chollet et al. Layer weight initializers. https://keras.io/api/layers/initializers/glorotuniform-class. Accessed: June 2021.

[7] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[8] Chamberlain Fong. Elliptification of Rectangular Imagery, 2019.

[9] Peter Gleeson. Escaping the Curse of Dimensionality. URL: https://www.freecodecamp.org/news/the-curse-of-dimensionality-how-we-can-save-big-data-from-itself-d9fa0f872335/, 2017. [Online; accesed on 26th of April].

[10] Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the international conference on Neural Networks*, volume 3, pages 11–14. IEEE Press New York, 1987.

[11] David Hilbert. Ueber die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[12] David Hilbert. Mathematische Probleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, pages 253–297, 1900.

[13] B. Igelnik and N. Parikh. Kolmogorov's spline network. *IEEE Transactions on Neural Networks*, 14(4):725–733, 2003.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017.

[15] A. N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR*, 114:953–956, 1957.

[16] Helge Kragh. The True (?) Story of Hilbert's Infinite Hotel, 2014.

[17] A Kupers. On space-filling curves and the Hahn-Mazurkiewicz theorem.

[18] V. Kurková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5:501–506, 1992.

[19] Shiyu Liang and Rayadurgam Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

[20] G. G. Lorentz. Metric Entropy, Widths, and Superpositions of Functions. *The American Mathematical Monthly*, 69(6):469–485, 1962.

[21] Guy MacDonald Morton. A computer oriented geodetic data base; and a new technique in file sequencing. 1966.

[22] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

[23] Tomaso Poggio and Federico Girosi. A Theory of Networks for Approximation and Learning. *Tech Rep A.I. Memo No. 1140*, 1140, 08 2001.

[24] Andrew Polar and Michael Poluektov. A deep machine learning algorithm for construction of the Kolmogorov–Arnold representation. *Engineering Applications of Artificial Intelligence*, 99:104137, 2021.

[25] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[27] A.J. Schmidt-Hieber. *Lecture Notes Mathematical Statistics*. 2020.

[28] A.J. Schmidt-Hieber. The Kolmogorov–Arnold representation theorem revisited. *Neural Networks*, 137:119–126, 2021.

[29] Sho Sonoda and Noboru Murata. Neural Network with Unbounded Activations is Universal Approximator. *CoRR*, abs/1505.03654, 2015.

[30] David Sprecher. On the Structure of Representations of Continuous Functions of Several Variables as Finite Sums of Continuous Functions of One Variable. *Proceedings of The American Mathematical Society - PROC AMER MATH SOC*, 17, 02 1966.

[31] David A Sprecher. An improvement in the superposition theorem of kolmogorov. *Journal of Mathematical Analysis and Applications*, 38(1):208–213, 1972.

[32] Henk van der Pol. ReLU-Network-for-KA-approximation. https://github.com/HenkvdPol/ReLU-Network-for-KA-approximation, 2021.