A FUNCTIONAL EQUATION WITH POLYNOMIAL SOLUTIONS AND APPLICATION TO NEURAL NETWORKS

Bruno Després, Matthieu Ancellin

A FUNCTIONAL EQUATION WITH POLYNOMIAL SOLUTIONS AND APPLICATION TO NEURAL NETWORKS

BRUNO DESPRÉS & MATTHIEU ANCELLIN

Abstract. We construct and discuss a functional equation with contraction property. The solutions are real univariate polynomials. The series solving the natural fixed point iterations have immediate interpretation in terms of Neural Networks with recursive properties and controlled accuracy.

1. Introduction

It has been observed recently in [14, 5] that a certain generalization of the Takagi function [10] to the square function $x \mapsto x^2$ has an interesting interpretation in terms of simple Neural Networks with the ReLU function $R(x) = \max(0, x)$ as an activation function. In [14], this generalization is the basis of a general theorem of approximation of functions by Neural Network architectures, see also [11]. We generalize the principle of the functional equation [9] to any real univariate polynomial $x \mapsto H(x)$, by using techniques which are standard in numerical analysis. In terms of the design and discussion of Neural Network architectures [8], the new formulas gain broader generality.

By considering the literature [8] on the current understanding of the mathematical structure of Neural Networks, the most original output of the construction is the novel functional equation with three main properties: a) it has general polynomial solutions under the conditions of the main Theorem, b) it is contractive, so is easily solved by any kind of standard fixed point procedure and, c) the converging fixed point iterations can be implemented as reference solutions in Feedforward Deep Networks with ReLU activation function [8][Chapter 6], with controlled accuracy. In other words, our construction associates a well posed functional equation and its solution to some simple Neural Networks. The proof that the equation has polynomial solutions is easy. It is possible that a similar construction has already been considered in the immense literature on polynomials but to our knowledge, never in combination with the discussion of Neural Networks architectures.

2. A contractive functional equation

The normalized closed interval is $I = [0, 1]$. The set of continuous functions $C^0(I)$ over I is equipped with the maximal norm $\|f\|_{L^\infty(I)} = \max_{i \in I} |f(x)|$.

Consider a subdivision in $m \geq 1$ subintervals $[x_j, x_{j+1}]$ where $0 = x_0 < x_1 < \cdots < x_j < \cdots < x_m = 1$, $x_j = jh$ and $h = 1/m$. Set

$$P^n = \{p \text{ real polynomial of degree } \leq n\}.$$

The set of continuous piecewise linear functions is

$$V_h = \left\{u \in C^0(I),\ u_{|(x_j, x_{j+1})} \in P^1 \text{ for all } 0 \leq j \leq m-1\right\}.$$

Similarly with the classical Finite Element setting [3], some basis functions are chosen in a subset of V_h, even if they are not basis functions in the classical Finite

2020 *Mathematics Subject Classification.* 65Q20, 65Y99, 78M32.

Keywords. Functional equation, numerical analysis, real polynomials, Neural Networks.

The authors thank CEA for support.

Element sense. In the proposed construction, they are taken in subset $E_h \subset V_h$

$$E_h = \{u \in V_h : \ u(I) \subset I, \ u \text{ is non constant on exactly one subinterval}\}.$$

The assumption $u(I) \subset I$ is critical to get the contraction property under the form of Lemma 2.5. Our interest in this set is because functions in E_h and V_h are easily assembled or implemented in Neural Networks with the ReLU function $R(x) = \max(0, x)$, see [8, 14, 5, 6, 11].

For the simplicity of the presentation, we start with a given real polynomial function $H \in P^n$. More general functions are discussed in the last section. We consider the problem below.

Problem 1. *Find* $(e_0, e_1, \ldots, e_r, \beta_1, \ldots, \beta_r) \in V_h \times (E_h)^r \times \mathbb{R}^r$ *such that the identity below holds*

$$H(x) = e_0(x) + \sum_{i=1}^{r} \beta_i H(e_i(x)), \quad x \in I, \tag{2.1}$$

with the contraction condition

$$K < 1, \qquad K = \sum_{i=1}^{r} |\beta_i|. \tag{2.2}$$

Because of the external composition by H in the last sum, the e_i's are not basis functions in the sense of the Finite Element Method [3]. Once the

$$(e_0, e_1, \ldots, e_r, \beta_1, \ldots, \beta_r) \in V_h \times (E_h)^r \times \mathbb{R}^r$$

are determined, equation (2.1) can be seen as a functional equation with H as a solution.

A first classical example is based on $H_1(x) = x(1-x)$ which satisfies [10, 14, 5]

$$H_1(x) = \frac{1}{4} g(x) + \frac{1}{4} H_1(g(x)) \tag{2.3}$$

where g is the hat function (normalized finite element function): $g(x) = 2x$ for $0 \le x \le \frac{1}{2}$ and $g(x) = 2(1-x)$ for $\frac{1}{2} \le x \le 1$. Set $e_1(x) = \min(2x, 1)$ and $e_2(x) = \min(2(1-x), 1)$ with $e_1, e_2 \in E_h$ for $h = 1/2$. One obtains

$$H_1(x) = e_0(x) + \frac{1}{4} H_1(e_1(x)) + \frac{1}{4} H_1(e_2(x)), \quad e_0(x) = \frac{1}{4}(g(x) - 1) \tag{2.4}$$

where the contraction property (2.2) is satisfied with a constant $\sum |\beta_i| = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$. Our second example concerns the function $H_2(x) = x^3$. Set $e_3(x) = 1 - e_2(x)$. One can check the formula

$$H_2(x) = e_0(x) + \frac{1}{8} H_2(e_1(x)) + \frac{1}{8} H_2(e_2(x)) + \frac{1}{4} H_2(e_3(x)) \text{ with } e_0(x) = \frac{3}{4} e_3(x) - \frac{1}{8}.$$

The condition of contraction is satisfied with the constant $1/2$.

Consider the case $m = 1$, that is just one subinterval, and take a polynomial H with $\deg(H) \ge 2$. For $H(x) = x^n +$ low order terms, then by equating the coefficients of x^n on both sides, one gets $1 = \sum \beta_i \mu_i^n$ with $\mu_i = e_i'(x) \in \mathbb{R}$. Because $e_i(I) \subset I$, then $|\mu_i| \le 1$. So $1 \le \sum |\beta_i| |\mu_i|^n \le \sum |\beta_i|$. It means the contraction condition (2.2) is not satisfied and Problem 1 has only trivial solutions for $m = 1$. Since the contraction property is crucial in the construction, we will not consider the case $m = 1$ anymore. The main result is below.

Theorem 2.1. *Let* $H \in P^n$. *There exists a threshold value* $m_*(H) \ge 2$ *such that the functional equation (2.1) has a solution with the contraction property (2.2) for all* $m \ge m_*(H) \Longleftrightarrow h \le 1/m_*(H)$.

Writing the basis functions as $e_i(x) = a_i + \frac{b_i - a_i}{h}(x - x_j)$ *in the subinterval where they are non constant, the parameters* (a_i, b_i) *can be taken as in Lemma 2.3 in the general case. They can also be taken as in Lemma 2.2 if* $H(x) = x^n$ *is a monomial.*

For $n \geq 2$, the number of basis functions which are non constant in a given subinterval is $n-1$, the basis functions are duplicated by translation from of subinterval to the other and so the total number of basis functions is $r = m(n-1)$.

The proof is based on decoupled and simpler problems posed in subintervals $[x_j, x_{j+1}]$. Let us note the second derivative of H as $p = H'' \in P^{n-2}$. The collection of reduced problems for all subinterval $[x_j, x_{j+1}]$ writes as follows.

Problem 2. *For all subintervals $0 \leq j \leq m-1$, find triples $(a_i, b_i, \gamma_i) \in I \times I \times \mathbb{R}$ $(1 \leq i \leq s)$ such that $b_i - a_i \neq 0$ for all i and*

$$p(x_j + hy) = \sum_{i=1}^{s} \gamma_i p(a_i + (b_i - a_i)y), \qquad y \in I. \tag{2.5}$$

Lemma 2.1. *The equation (2.1) is equivalent to the equation (2.5).*

Proof. The proof is in two parts.
(2.1) $\Rightarrow$ (2.5): on the interval $[x_j, x_{j+1}]$, one can write $e_i(x) = a_i + \frac{b_i - a_i}{h}(x - x_j)$. By differentiation, a solution to (2.1) gives $p(x) = \sum_{i=1}^{r} \beta_i \left(\frac{b_i - a_i}{h}\right)^2 p(e_i(x))$ for $x_j < x < x_{j+1}$. Retaining in the sum only the indices i such that $b_i - a_i \neq 0$, one gets (2.5) where $x = x_j + hy$ and $e_i(x) = a_i + (b_i - a_i)y$.
(2.5) $\Rightarrow$ (2.1): rewrite the discrete quantities in (2.5) with another lower index j which refers to the interval in which this equation is considered. It defines $a_{i,j}$, $b_{i,j}$ and $\gamma_{i,j}$. Define

$$e_{i,j}(x) = \begin{cases} a_{i,j} & \text{for } 0 \leq x \leq x_j, \\ a_{i,j} + \frac{b_{i,j} - a_{i,j}}{h}(x - x_j) & \text{for } x_j \leq x \leq x_{j+1} = x_j + h, \\ b_{i,j} & \text{for } x_{j+1} \leq x \leq 1. \end{cases}$$

Define also

$$\beta_{i,j} = \frac{h^2}{(b_{i,j} - a_{i,j})^2} \gamma_{i,j}, \qquad \text{where } b_{i,j} - a_{i,j} \neq 0. \tag{2.6}$$

Consider the function

$$e_0(x) = H(x) - \sum_j \sum_i \beta_{i,j} H(e_{i,j}(x)). \tag{2.7}$$

By construction e_0 is continuous and its second derivative is zero in all open subintervals (x_j, x_{j+1}). Therefore $e_0 \in V_h$ which ends the proof. □

If the polynomials $y \mapsto p(a_{i,j} + (b_{i,j} - a_{i,j})y)$, $1 \leq i \leq s$, generate a complete system in P^{n-2}, then the equation (2.5) has a solution. That is why we will consider from now on that

$$s = \dim(P^{n-2}) = n - 1. \tag{2.8}$$

Next, by differentiation, the equation (2.5) in $[x_j, x_{j+1}]$ is equivalent to the square linear system

$$M_j X_j = b_j, \qquad 0 \leq j \leq m-1. \tag{2.9}$$

The square matrix is

$$M_j = \begin{pmatrix} p(a_{1,j}) & p(a_{2,j}) & \dots & p(a_{n-1,j}) \\ c_{1,j} p'(a_{1,j}) & c_{2,j} p'(a_{2,j}) & \dots & c_{n-1,j} p'(a_{n-1,j}) \\ \dots & \dots & \dots & \dots \\ c_{1,j}^{n-2} p^{(n-2)}(a_{1,j}) & c_{2,j}^{n-2} p^{(n-2)}(a_{2,j}) & \dots & c_{n-1,j}^{n-2} p^{(n-2)}(a_{n-1,j}) \end{pmatrix} \in \mathcal{M}_{n-1}(\mathbb{R}) \tag{2.10}$$

where we note $c_{i,j} = b_{i,j} - a_{i,j}$. The unknown of the linear system is $X_j = (\gamma_{1,j}, \gamma_{2,j}, \dots, \gamma_{n-1,j})^T \in \mathbb{R}^{n-1}$. The right hand side of the linear system is $b_j = \big(p(x_j), hp'(x_j), \dots, h^{n-1} p^{(n-1)}(x_j)\big)^T \in \mathbb{R}^{n-1}$ which is bounded $\|b_j\|_\infty \leq C$ uniformly with respect to the subinterval index j.

One remarks that: a) the matrix M_j is close to a Vandermonde matrix, so natural invertibility conditions arise; b) provided the real numbers $a_{i,j}, b_{i,j} \in [0,1]$ are chosen independently of the subinterval (it will be written $a_{i,j} = a_i$ and $b_{i,j} = b_i$), then $M_j = M$ is independent of the index j. Two cases of invertibility and one case of non invertibility are considered below.

Lemma 2.2. *Take $p(x) = x^{n-2}$ with $n-2 \geq 0$. Assume the real numbers $a_{i,j} = a_i \in [0,1]$ and $b_{i,j} = b_i \in [0,1]$ are chosen independently of the subinterval (so $M_j = M$ is independent of the index j) and $b_i - a_i \neq 0$ for all i. Then M is non singular if and only if $a_i b_k - a_k b_i \neq 0$ for all $1 \leq i \neq k \leq n-2$.*

Proof. The matrix is $M = \left(\frac{n!}{(n-t)!} (b_i - a_i)^t a_i^{n-t} \right)_{1 \leq t+1, i \leq n-1}$. By assumption $b_i - a_i \neq 0$ for all i, so M is similar to $N = \left(\left(\frac{a_i}{(b_i - a_i)} \right)^{n-t} \right)_{1 \leq t+1, i \leq n-1}$. It is a Vandermonde matrix, invertible if and only if $\frac{a_i}{b_i - a_i} \neq \frac{a_k}{b_k - a_k}$ for $i \neq k$. The latter condition is equivalent to $a_i b_k - a_k b_i \neq 0$. □

Lemma 2.3. *Take $p \in P^{n-2}$ with $deg(p) = n-2 \geq 0$. Assume the real numbers $a_{i,j} = a_i \in [0,1]$ and $b_{i,j} = b_i \in [0,1]$ are chosen independently of the subinterval. Assume $a_i \neq a_k$ and $b_i - a_i = b_k - a_k \neq 0$ for all $1 \leq i \neq k \leq n-2$. Then the matrix M is non singular.*

Proof. The matrix M is similar to the matrix $N = \left(p^{(t)}(a_i) \right)_{1 \leq t+1, i \leq n-1}$ which is a reducible to a non singular Vandermonde matrix. □

Lemma 2.4. *Take $p(x) = u + x$, $e_1(x) = a_1 + (b_1 - a_1)x$ and $e_2(x) = a_2 + (b_2 - a_2)x$. Then the matrix M is singular if and only if $u(b_2 - a_2 - b_1 + a_1) + a_1 b_2 - a_2 b_1 = 0$.*

Proof. Indeed $p(e_1(x)) = u + a_1 + (b_1 - a_1)x$ and $p(e_2(x)) = u + a_2 + (b_2 - a_2)x$. The condition of linear independence of these two linear polynomials reduces to the claim. □

Proof of Theorem 2.1. If $n = 0$ or $n = 1$ the result is trivial, so we consider $n - 2 \geq 0$. If $H(x) = x^n$ is a monomial function, one can takes the first set of basis functions given by Lemma 2.2 because the matrices are non singular. Unfortunately, this simple choice is not always possible as shown by Lemma 2.4. So to cover the case of general polynomials $H \in P^n$, we continue with basis functions satisfying Lemma 2.3.

One notes $\mu = \frac{1}{2(n-1)}$. In a generic subinterval , we construct functions e_i for $1 \leq i \leq n-1$ by taking $a_i = i\mu$ and $b_i = (i+1)\mu$. By construction $b_i - a_i = b_k - a_k = \mu > 0$, $0 \leq a_i \leq 1$, $0 \leq b_i \leq \frac{n}{2(n-1)} \leq 1$ (because $n \geq 2$) and $a_i \neq a_k$ for $i \neq k$.

The matrix $M_j = M$ being non singular, then the system (2.9) has a solution $X_j = (\gamma_{1,j}, \ldots, \gamma_{n-1,j})$ such that $\|X_j\|_\infty \leq \|M^{-1}\|_\infty \|b_j\|_\infty \leq C$ uniformly with respect to the index of the subinterval j. So the representation (2.7) of H holds for $x \in I$.

The constant is

$$K = \sum_{j=0}^{m-1} \sum_{i=1}^{n-1} |\beta_{i,j}| \leq \sum_{j=0}^{m-1} \sum_{i=1}^{n-1} \frac{h^2 |\gamma_{i,j}|}{(b_i - a_i)^2} \leq \sum_{j=0}^{m-1} \sum_{i=1}^{n-1} \frac{h^2 C}{\mu^2} \leq m(n-1) \frac{h^2 C}{\mu^2} = \frac{(n-1)C}{\mu^2 m}.$$

Take $m^*(H) > \frac{(n-1)C}{\mu^2}$. So the contraction property is satisfied for $m \geq m^*(H)$. □

Lemma 2.5. *Under the contraction condition (2.2), one has the bounds $\|H\|_{L^\infty(I)} \leq \frac{1}{1-K} \|e_0\|_{L^\infty(I)}$ and $\| \sum_{i \geq 1} \beta_i H \circ e_i \|_{L^\infty(I)} \leq \frac{K}{1-K} \|e_0\|_{L^\infty(I)}$.*

Proof. It is evident but we detail it because the key condition $e_i(I) \subset I$ is used. Consider the linear operator

$$\begin{array}{rccc} \mathcal{H}: & L^\infty(I) & \longrightarrow & L^\infty(I) \\ & G & \longmapsto & \sum_{i\geq 1} \beta_i G \circ e_i. \end{array} \tag{2.11}$$

Then $\|\mathcal{H}\|_{\mathcal{L}(L^\infty(I))} \leq K < 1$, that is $\mathcal{H}$ is a strictly contractive operator. The functional equation rewrites $H = e_0 + \mathcal{H}(H)$ from which the inequalities are deduced. □

3. Application to Neural Networks

In this section, we detail some algorithms which have a natural interpretation in the language of Neural Networks with the ReLU function as an activation function [5, 14, 8]. These algorithms provide reference solutions based on the standard fixed point method where the iteration index is $k = 0, 1, \ldots$

$$\left\{ \begin{array}{l} H_0 = 0, \\ H_{k+1} = e_0 + \displaystyle\sum_{1\leq i\leq r} \beta_i H_k \circ e_i. \end{array} \right. \tag{3.1}$$

The first terms of the series are $H_1 = e_0$, $H_2 = e_0 + \sum_i \beta_i e_0 \circ e_i$, $H_3 = e_0 + \sum_i \beta_i e_0 \circ e_i + \sum_{i,j} \beta_i \beta_j e_0 \circ e_i \circ e_j$ and more generally

$$H_k = e_0 + \sum_{p=1}^{k-1} \left(\sum_{1\leq i_1,\ldots,i_p\leq r} \left(\beta_{i_1} \ldots \beta_{i_p}\right) e_0 \circ e_{i_1} \circ \cdots \circ e_{i_p} \right), \quad k \geq 1. \tag{3.2}$$

Lemma 2.5 and the contractive operator (2.11) yield a standard convergence property with exponential rate in $L^\infty(I)$

$$\|H_k - H\|_{L^\infty(I)} \leq K^k \|H\|_{L^\infty(I)}, \qquad K = \sum_{1\leq i\leq r} |\beta_i| < 1. \tag{3.3}$$

3.1. A first Neural Network implementation.

To explain how the function H_k can be implemented in a Neural Network, we need more notations. The ReLU function is denoted as $R(x) = \max(0, x)$. The ReLU function T with threshold is denoted as [1]

$$T(x) = \max(0, \min(x, 1)), \qquad \text{with } T(x) = \min(R(x), 1) = R(x) - R(x-1). \tag{3.4}$$

In the language of Neural Networks [8], functions R and T are called activation functions. The function T is clearly well adapted to encode the functions in E_h, that is why we describe in details some implementations features with this function. Using (3.4), these implementations can be performed without any difficulty with the ReLU function R.

It is convenient for the rest of the discussion to define a set $\mathcal{E}$ more general than E_h which is included in the new set $\mathcal{E}$

$$\begin{array}{rl} \mathcal{E} = \big\{ e \in C^0(I): & \text{there exists } 0 \leq \alpha < \beta \leq 1 \text{ and } 0 \leq a \leq b \leq 1 \text{ such that} \\ & e(x) = a \text{ for } 0 \leq x \leq \alpha, \\ & e(x) = a + (b-a)\frac{x-\alpha}{\beta-\alpha} \text{ for } \alpha \leq x \leq \beta, \\ & e(x) = b \text{ for } \beta \leq x \leq 1 \big\}. \end{array} \tag{3.5}$$

We will also use the standard notation $Lx = a + bx$ for affine functions where $a, b \in \mathbb{R}$. Same notations hold for $L_1 x = a_1 + b_1 x$, and so on. We immediately remark that the composition of two affine functions is also an affine function, that is with natural notations $L_1 \circ L_2 = L_3$.

[1] This function is a variant of the function hard-tanh [8]. See also the documentation `https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu`.

Lemma 3.1. *All functions $e \in \mathcal{E}$ are computable with a composition of, first of all an affine function, next the activation function T and finally an affine function (that is $e = L_1 \circ T \circ L_2$).*

Proof. Indeed, with the notation of (3.5), $e(x) = a+(b-a)T\left((x-\alpha)/(\beta-\alpha)\right)$. □

Lemma 3.2. *The series (3.2) can be implemented by linear combination of functions which are the result of at most k activation function T composed with at most $k+1$ affine functions.*

Proof. The basic operations in a Neural Network are composition and linear combination of functions, as sketched in Figure 3.1. These operations are already sufficient to implement the iterations (3.1).

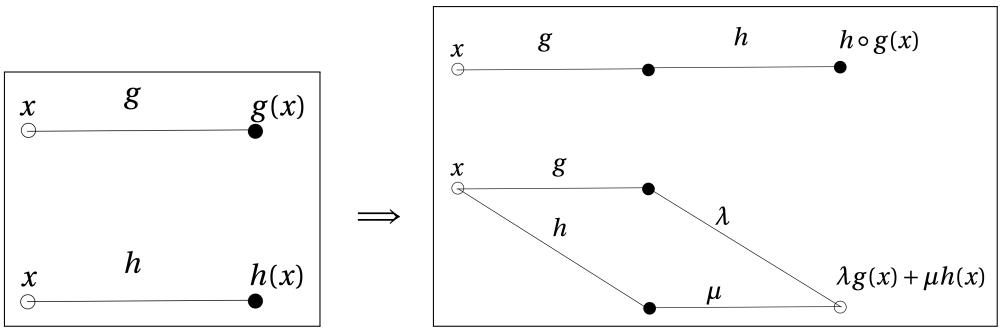

FIGURE 3.1. Sketch in a Neural Network of the composition $h \circ g$ and of the linear combination $\lambda g+\mu h$ of 2 functions g and h already implemented as Neural Networks. Empty bullets correspond either to the input or to pure linear combination. Black bullets show that an activation function R, T or another one is used.

Now consider (3.2). Since $e_0 \in V_h$, then it is also a affine combination of function in E_h. Therefore $e_0 \circ e_{i_1} \circ \cdots \circ e_{i_k}$ can be assembled by linear combination of composition of the activation functions T and affine functions, with at most k activation functions T and at most $k+1$ affine functions (each of them obtained by the composition of at most 2 affine functions). □

So the structure described by the Lemma can be implemented in a Neural Network with activation T (or R). It generalizes to any polynomial the algorithm presented in [5] or used in [6]. The depth of the network is the number of composition of activation functions, so the depth is k. This implementation is a Neural Network generalization to all polynomials H of the series in [5, 14, 8] for the polynomial $x \mapsto x - x^2$.

3.2. **Accuracy.** Considering (3.3), two ways to obtain a better accuracy are either to increase m, that is to increase the number of neurons in the first layer, or to increase k, that is to increase the number of layers.

The series (3.2) truncated at $k = 0$ recovers the polynomial H with an accuracy which arbitrarily small because $K = O(h)$: the number of neurons is $r = ms = s/h = O(h^{-1})$. Since polynomial functions are dense in the space $C^0(I)$ in the maximum norm, it can be rephrased as a new constructive proof in dimension one of the Cybenko Theorem [4]. Using the language of Neural Networks, one hidden layer of an arbitrary large number of neurons can approximate any function in $C^0(I)$.

Another strategy is to increase the number of layers $\mathbf{k} = \mathbf{1}, \mathbf{2}, \mathbf{3}, \dots$ in the series (3.2). The structure of the direct implementation of such a Neural Network is non standard with respect to the literature [8] because the width of the layers has a dependence with respect to k proportional to r^k which is the numbers of terms in the series (3.2): due to additional overhead, the cost of the implementation or the run is more $O\left((r+c)^k\right)$; the approximation of the function e_0 brings another cost. However the accuracy is also a power of k, see (3.3), so there is a balance between the cost and the accuracy.

3.3. **Damping the width.** It is possible to circumvent the exponential growth of the width of the layers by two techniques which are explained below. This is what we call damping the growth of the width, or damping the width.

3.3.1. *Splitting strategy.* Here we use a time interpretation of the fixed procedure coupled with a splitting procedure. It is based on the ordinary differential equation for the function $G(t) \in L^\infty(I)$

$$\begin{cases} G(t) = 0, \\ G'(t) = e_0 + \displaystyle\sum_{1 \le i \le r} \beta_i G(t) \circ e_i - G(t). \end{cases} \tag{3.6}$$

One has the bound $\|G(t) - H\|_{L^\infty(I)} \le e^{-(1-K)t} \|H\|_{L^\infty(I)}$. A possible splitting technique is based on decreasing time steps $\Delta t_1 \ge \cdots \ge \Delta t_k \ge \Delta t_{k+1}$ such that the time step tends to 0 ($\Delta t_k \to 0$) and the total time tends to infinity ($t_k = \sum_{l=1}^k \Delta t_l \to \infty$). Next within one time step $t_k \le t \le t_{k+1} = t_k + \Delta t_{k+1}$, one may consider the series of ODEs

$$\begin{cases} G_0'(t) = e_0 - (1 - \sum_{i=1}^r |\beta_i|) G(t), & 0 \le t \le \Delta t_k \\ G_i'(t) = \beta_i G_i(t) \circ e_i - |\beta_i| G_i(t), & 0 \le t \le \Delta t_k, \quad \text{for } i = 1, \dots, r, \end{cases}$$

where $G_0(0) = G(t_k)$, $G_{i+1}(0) = G_i(\Delta t_k)$ for $0 \le i \le r-1$ and finally $G(t_{k+1}) = G_r(\Delta t_k)$.

A fully discrete version of the method takes the form

$$\begin{cases} \frac{G_{k+1/r} - G_k}{\Delta t_k} = e_0 - (1 - \sum_{i=1}^r |\beta_i|) G_k, \\ \frac{G_{k+(i+1)/r} - G_{k+i/r}}{\Delta t_k} = \beta_i G_{k+i/r} \circ e_1 - |\beta_i| G_{k+i/r} \quad \text{for } i = 1, \dots, r. \end{cases}$$

At each step of the algorithm, all functions can be updated with the two operations, composition and linear combination, described in Figure 3.1. Natural bounds can be written to estimate $\|G_k - H\|_{L^\infty(I)}$. This method damps the exponential growth of the width, because the steps generate a series similar to the general one (3.2) but with less terms in the right hand side. Nevertheless this gain is mitigated by the number of additional fractional steps.

3.3.2. *Reconfiguration of the Network.* Independently of the method by splitting explained just above, there is the possibility to use the main Theorem of [5] which explains that a function f which is piecewise linear in $[0,1]$ can be implemented in a Neural Network with the ReLU function R with a depth (the number of layers) proportional to the number of breakpoints of f and an arbitrary constant width (the number of neurons per layer) $W \ge 4$: it can be called a reconfiguration of the Network because the operations needed to describe a piecewise function f with a certain number of breakpoints consists in an explicit rearrangement of the order with which the calculations are done.

In what follows we briefly give a new proof of this result for the function that corresponds to H_k in (3.2) and to the activation function T. It yields reconfiguration of the Neural Network and damps the width of the layers. This proof has its own

interest because the width of the new resulting Network $W = 3$ is smaller than the one made explicit in [5] which is $W \geq 4$.

One starts with a preliminary Lemma which shows that the composition of T, next an affine function and finally T can be expressed as the composition of an affine function, next T and finally an affine function.

Lemma 3.3. *Take two parameters $\lambda, \mu \in \mathbb{R}$. One has the identity*

$$T(\lambda T(x) + \mu) = T(\mu) + \lambda(M - m)T\left(\frac{x - m}{M - m}\right), \qquad x \in I,$$

where $m = \min\left(T\left(\frac{1-\mu}{\lambda}\right), T\left(\frac{-\mu}{\lambda}\right)\right)$ and $M = \max\left(T\left(\frac{1-\mu}{\lambda}\right), T\left(\frac{-\mu}{\lambda}\right)\right)$.

Proof. In the calculations below, all functions are continuous with respect to variables (x, λ, μ) and have a derivative almost everywhere with respect to the variable x.
One has $\frac{d}{dx}T(\lambda T(x) + \mu) = \lambda T'(x)T'(\lambda T(x) + \mu)$. So the derivative is non zero if and only if

$$\lambda \neq 0,\ 0 < x < 1,\ 0 < \lambda T(x) + \mu < 1 \Leftrightarrow \lambda \neq 0,\ 0 < x < 1,\ 0 < \lambda x + \mu < 1.$$

Assume $\lambda > 0$. The last inequality is $-\frac{\mu}{\lambda} < x < \frac{1-\mu}{\lambda}$. One gets $0 < x < 1$ with $-\frac{\mu}{\lambda} < x < \frac{1-\mu}{\lambda} \Leftrightarrow T\left(\frac{-\mu}{\lambda}\right) < x < T\left(\frac{1-\mu}{\lambda}\right)$. For $\lambda < 0$, the bounds are reversed, one finds the condition $T\left(\frac{1-\mu}{\lambda}\right) < x < T\left(\frac{-\mu}{\lambda}\right)$.
In summary the function $x \mapsto T(\lambda T(x)+\mu)$ has a zero derivative almost everywhere except if $m < x < M$. For x in this interval, the derivative is equal to λ.

Consider the function defined by $G(x) = \lambda(M - m)T\left(\frac{x-m}{M-m}\right)$. In the case $M - m > 0$, the function G has a zero derivative almost everywhere except if $m < x < M$, and then its derivative is equal to λ by a direct calculation. Therefore the difference $T(\lambda T(x) + \mu) - G(x) = C(\lambda, \mu)$ is constant with respect to x. In the case $M - m = 0$, one checks directly that the difference is also constant with respect to x.

It remains to identify the constant. Take $M - m > 0$ and let $x \to -\infty$. One gets the identity $T(\mu) - 0 = C(\lambda, \mu)$. In the case $M - m = 0$, the result is the same, and the proof is ended. □

Proposition 3.1. *One has $\mathcal{E} \circ \mathcal{E} = \mathcal{E}$.*

Proof. Take two general functions $e_1, e_2 \in \mathcal{E}$. Let us write $e_1 = L_1 \circ T \circ L_2$ and $\mathbf{e}_2 = L_3 \circ T \circ L_4$ where $L_{1,2,3,4}$ are affine functions. Since the composition of affine functions is an affine function, one can write

$$e_1 \circ e_2 = L_1 \circ T \circ (L_2 \circ L_3) \circ ToL_4 = L_1 \circ (ToL_5 \circ T) \circ L_4.$$

Lemma 3.3 shows there exists two affine functions L_6 and L_7 such that $T \circ L_5 \circ T = L_6 \circ T \circ L_7$. We note if $M - m = 0$ (resp. $\lambda = 0$) in Lemma 3.3, then the singularity $\frac{1}{M-m}$ (resp. $\frac{1}{\lambda}$) is meaningless because of the exterior multiplication by $M - m = 0$ (resp. $\lambda = 0$). So in this case these singularities are artificial and L_7 can be chosen arbitrarily. One gets

$$e_1 \circ e_2 = L_1 \circ L_6 \circ T \circ L_7 \circ L_4 = L_8 \circ T \circ L_9.$$

This function has the generic form of functions in $\mathcal{E}$ because it is the composition of an affine function, next T, and finally an affine function. It shows that $e_1 \circ e_2 \in \mathcal{E}$. So $\mathcal{E} \circ \mathcal{E} \subset \mathcal{E}$.

Finally take $e_2(x) = x$. Then $e_1 = e_1 \circ e_2$ so $\mathcal{E} \subset \mathcal{E} \circ \mathcal{E}$. Therefore the equality of the claim holds. □

Consider the function H_k in (3.2). Expansion of e_0 as a linear combination of functions in E_h and repeated application of Proposition 3.1 show that the sum (3.2) can be reorganized as

$$H_k = \sum_{d=1}^{D} L_d^+ \circ T \circ L_d^- \tag{3.7}$$

where $L_d^{\pm}$ are affine functions and D is taken large enough, equal to the total number of terms in (3.2). In virtue of the contraction condition, the series is convergent $\sum_{d=1}^{D} \left\| L_d^+ \circ T \circ L_d^- \right\|_{L^\infty(I)} < \infty$.

Theorem 3.1. *The series (3.7) can be implemented within a Neural Network with the ReLU function with T as activation function. The width is $W = 3$ and the depth is D.*

Proof. The proof is based on two ideas. The first idea is an iterative calculation of the result. The second idea comes from [5] and the implementation uses a source channel and a collation channel. Preliminary to describing the implementation, we rewrite the series (3.7) as

$$H_k = C\left(\frac{1}{2} + \frac{1}{C}\sum_{d=1}^{D} L_d^+ \circ T \circ L_d^-\right) - \frac{C}{2}$$

where $C \geq 2\sum_{d=1}^{D} \left\| L_d^+ \circ T \circ L_d^- \right\|_{L^\infty(I)}$ is taken sufficiently large.

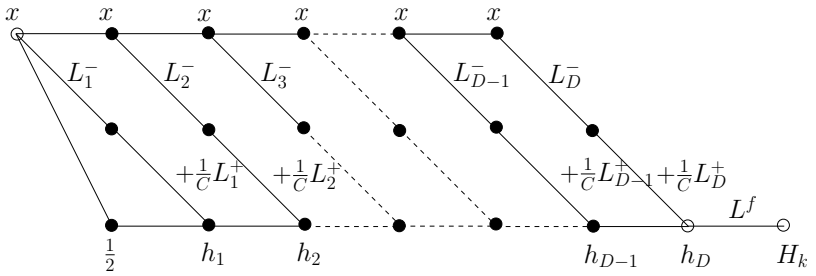

FIGURE 3.2. Structure of the Neural Network associated to the formula (3.7) and to the iterations (3.8). The top line reproduces x, it is called a source channel. The bottom line is the collection channel which realizes the addition in (3.8), it is called a collation channel. The intermediate lines correspond to the calculation of the different affine functions. The black bullets are the Neurons, they represent the application of the ReLU function with threshold T. The Neurons leave unchanged terms in the source channel and in the collation channel. The three empty bullets at the input and at the output signal that no ReLU is applied.

Then partial series $h_r = \frac{1}{2} + \frac{1}{C}\sum_{d=1}^{r} L_d^+ \circ T \circ L_d^-$ (for $1 \leq r \leq D$) take values between 0 and 1 (for $x \in I$), that is $h_r(I) \subset I$ and $Toh_r = h_r$. One notes the recurrence relation

$$h_0 = \frac{1}{2}, \quad h_{r+1} = h_r + \frac{1}{C} L_{r+1}^+ \circ T \circ L_{r+1}^- \quad 1 \leq r \leq D-1. \tag{3.8}$$

The final result is obtained which a last affine function $H_k = Ch_D - \frac{C}{2} = L^f \circ h_D$. This recurrence is easily implemented in accordance with the diagram described in Figure 3.2, using a source channel and a collation channel as it is explained in [5]. The maximal number of Neurons per layer is $W = 3$.

The depth is the number of hidden layers between the input layer and the output layer. In the Figure 3.2 it is equal to $D+1$. However it is possible to concatenate the last step with the previous one because only affine functions are involved and the composition of two affine functions can be implemented as just one affine function. It saves one hidden layer, so the depth can be made equal to D. □

In summary the function H_k can be calculated either from the formula (3.2) which yields $k+1$ layers with an increasing number of neurons per layers, or with the Network that comes from Theorem 3.1 which has a large number of layers and 3 neurons per layer. One can also implement (3.7) directly with exactly one hidden layer and with one input layer and one output layer. Many other intermediate Networks, in terms of the number of layers and neurons per layer, are possible.

3.3.3. *Recursive and recurrent Neural Networks.* It will be valuable in a near future to compare with the structure of some Neural Networks which have an iterative nature. In recursive Neural Networks, similar weights are used in consecutive layers [8]. In recurrent Neural Networks [2, 8] which are widely used for time signals analysis, some level of recursivity is also introduced but in a different way in order to propose an efficient treatment of time series.

3.4. **Numerical examples.** It is possible to calculate analytically all coefficients β_i and γ_i by solving the linear system (2.9), and to compare numerically the numerical error with the theoretical rate of convergence (3.3), as done in Table 3.4. Since it is elementary, we complement with another more interesting approach, which is to evaluate the efficiency of the training of the coefficients for a certain depth k and a certain collection of basis functions e_i for $1 \leq i \leq r$, with r conveniently chosen. We formulate training as the problem of finding the best coefficients β_i for $1 \leq i \leq r$ and $e_0 \in V_h$ by minimizing the L^2 cost function

$$W = (e_0, \beta_1, \ldots, \beta_r) \mapsto J(W) = \|H_k(W) - H\|_{L^2(I)}\,, \tag{3.9}$$

where the integral (in the L^2 norm) is evaluated by quadrature and the basis functions e_i for $i \geq 1$ are chosen in advance. This procedure provides insights in the convergence properties of the training stage which is a major algorithmic issue [8] for Neural Networks. For exemple, in [6], various tests for $x \mapsto x^2$ implemented within a dense Neural Network of width 3 and arbitrary depth k completely failed to recover the accuracy $O(4^{-k})$. In the tests below which are based on the structure explained in this document, the asymptotic accuracy is much better captured.

Elementary tests were implemented[2] in Julia (see `https://julialang.org`) using an automatic differentiation library [13] and an optimization library [12]. The integral in (3.9) being evaluated by quadrature with uniform discretization, it is equivalent to say that the dataset for training is

$$\mathcal{D} = \{(x_i, H(x_i))\,, i = 1, 2, \ldots, N\}$$

where $0 \leq x_i = \frac{i-1}{N-1} \leq 1$. Since we take $N = 1000$ in our tests, the dataset is oversampled. We used various standard optimizers, such as the Newton-Raphson method, the LBFGS quasi-Newton method or a simple Gradient Descent method with line research. Tests with the Gradient Descent showed that the optimum is unchanged but the rate of convergence is much slower. We expect that with Stochastic Gradient Descent [8], the convergence will be even slower.

So to display results with good accuracy obtained in a reasonnable time, we present the results obtained with the Newton or quasi-Newton method. We record the accuracy in function of the depth k. For increasing values of k, we report the

[2]Source code is available at `https://doi.org/10.5281/zenodo.3936433`. It implements the iterations (3.1) with the two operations described in Figure 3.1.

numerical $L^2(I)$ error $E(k) = J(W_k)$ where the optimal value W_k has been obtained at the end of the training and $K(k)$ which is the value of the constant K evaluated in function of W_k. A value $K(k) < 1$ is an indication of the stability of the method and can be compared with the theoretical value (see Table 3.4).

For the tests in Tables 3.1 and 3.2, the basis functions are $e_1(x) = \min(2x, 1)$, $e_2(x) = \max(1 - 2x, 0)$, $e_3(x) = \max(0, 2x - 1)$ and $e_4(x) = \min(1, 2 - 2x)$. By comparison with the tests shown in [6], the gain in accuracy as a function of k (and of stability) is spectacular, even if these new tests need much more neurons per layers for large k. These first two examples show the scaling $E(k) = O(\alpha^k)$ with $\alpha \approx \frac{1}{4} < \min_{2\leq k\leq 8} K(k)$, that is the convergence is at a better rate than what is predicted by the evaluation of $K(k)$. This is confirmed by the other examples below.

Next we perform similar tests but with $m = 3$ subintervals. The accuracy is better than for $m = 2$ subintervals. The factor per layer is $\approx 1/9$ (it comes probably from the formula $x^2 = e_0(x) + \frac{1}{9}g(x)^2$ which is easy to check). The numerical value of $K(k)$ is close to $1/3$.

k	1	2	3	4	5	6	7	8
$E(k)$	1.8e-2	4.6e-3	1.2e-3	2.9e-4	7.3e-5	1.8e-5	4.6e-6	1.1e-6
$K(k)$	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5

TABLE 3.1. The polynomial is $H(x) = x - x^2$. One observes $E(k) = O(4^{-k})$ in accordance with the theoretical prediction issued from (2.3).

k	1	2	3	4	5	6	7	8
$E(k)$	2.5e-2	6.2e-3	1.5e-3	3.9e-4	9.6e-5	2.4e-5	6.0e-6	1.5e-6
$K(k)$	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5

TABLE 3.2. The polynomial $H(x) = x - x^2/2 - x^3/2$. One also observes $E(k) = O(4^{-k})$.

k	1	2	3	4
$E(k)$	3.3e-2	3.4e-3	3.8e-4	4.2e-5
$K(k)$	0	0.94	0.38	0.36

TABLE 3.3. The polynomial is $H(x) = x - x^2$. One observes $E(k) \approx O(9^{-k})$ in accordance with the best theoretical prediction.

In the last test, we take $H(x) = x^2 + x^3 + x^4$, $m = 3$ subintervals and 9 basis functions in total. The three basis functions $e_1(x) = \min(3x, 1)$, $e_2(x) = \max(1 - 3x, 0)$ and $e_3(x) = \min(1/3 + 2x, 1)$ are duplicated by translation in the three subintervals. The results are in Table 3.4, where we also provide the accuracy with the Neural Network with the exact coefficients (obtained by solving the linear systems (2.9)) and the exact constant K. One observes convergence of the training at a better rate than the theoretical prediction.

Other tests have been made. Two difficulties were observed: the constant $K(k)$ can be greater than one as in Table 3.4, which makes the results more difficult to interpret; or the time of training with a large number of basis functions (≥ 10) becomes important with our current implementation.

k	1	2	3	4
$E(k)$	4.4e-2	4.5e-3	6.0e-4	2.7e-4
$K(k)$	0	3.53	0.39	0.46
$E_{\rm ex}(k)$	8.4e-1	6.3e-1	4.6e-1	3.3e-1
$K_{\rm ex}(k)$		0.94	0.94	0.94

TABLE 3.4. The function is $H(x) = x^2+x^3+x^4$ and the number of subintervals is $m = 3$. One observes good initial convergence, and low gain of accuracy from $k = 3$ to $k = 4$ layers. It is correlated to large value of $K(4)$. The trained solution is better than the exact solution: $E(k) \leq E_{\rm ex}(k)$.

4. Last remarks

Firstly we think it is worthwhile to replace the set E_h by the set

$$F_h = \{u \in V_h : \ u(I) \subset I\} .$$

Considering (2.3), we conjecture that the contraction constant $\sum |\beta_i|$ will be better with F_h instead of E_h: indeed the contraction constant is $1/4$ in (2.3), better than $1/2$ in (2.4) (with this respect, the set E_h is non optimal). Instead of matrices like (2.10) local to the subintervals, one will get a global matrix coupling all subintervals. The analysis of the global matrix is still to be done.

Secondly an interesting issue would be to replace the polynomial H by piecewise polynomial continuous functions. It would make strong connections with high order finite elements [3].

Thirdly, new questions arise for multivariate versions of Problem 1 because the theory of multivariate polynomials is more involved than univariate polynomials and, if one follows a similar strategy as the one presented. the matrices will necessarily be global.

Fourthly, the exact solutions obtained from (3.2) can be used for evaluation (benchmarking) of various Neural Networks implementations with ReLU activation functions.

Fifthly, it is possible to use our approach to modify a key step in modern proofs of convergence [5, 14, 11] of Deep Neural Networks with ReLU functions. This is left for further research.

Last, we point out the recent work [1] where a fixed point equation is also introduced in relation with a Neural Network architecture, but for a different purpose.

References

[1] A. Bensoussan, Y. Li, D. P. C. Nguyen, M.-B. Tran, S. C. P. Yam, X. Zhou, "Machine Learning and Control Theory", arXiv:2006.05604.

[2] M. Bodèn, "A Guide to Recurrent Neural Networks and Backpropagation", in *the Dallas project, SICS technical report T2002:03*, SICS, 2002.

[3] P. G. Ciarlet, *Linear and nonlinear functional analysis with applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Math. Control Signals Systems* **2** (1989), no. 4, p. 303-314.

[5] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, G. Petrova, "Nonlinear Approximation and (Deep) ReLU Networks", arxiv:905:02199v1.

[6] B. Després, "Machine Learning, adaptive numerical approximation and VOF methods", 2020, colloquium LJLL/Sorbonne university, `https://www.youtube.com/watch?v=OPKFYe01hH4`.

[7] B. Després, H. Jourdren, "Machine Learning design of Volume of Fluid schemes for compressible flows", *Journal of Computational Physics* **408** (2020).

[8] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[9] M. Hata, M. Yamaguti, "Weierstrass's function and chaos", *Hokkaido Mathematical Journal* **12** (1983), p. 333-342.
[10] ———, "The Takagi Function and Its Generalization", *Japan J. AppL Math.* **1** (1984), p. 83-199.
[11] J. Lu, Z. Shen, H. Yang, S. Zhang, "Deep Network Approximation for Smooth Functions", `https://blog.nus.edu.sg/matzuows/publications/`, 2020.
[12] P. K. Mogensen, A. N. Riseth, "Optim: A mathematical optimization package for Julia", *Journal of Open Source Software* **3** (2018), no. 24, p. 615, `https://doi.org/10.21105/joss.00615`.
[13] J. Revels, M. Lubin, T. Papamarkou, "Forward-Mode Automatic Differentiation in Julia", *arXiv:1607.07892* (2016).
[14] D. Yarotsky, "Error bounds for approximations with deep ReLU networks", *Neural Networks* **97** (2017), p. 103-114.

Laboratoire Jacques-Louis Lions, Sorbonne Université, 4 place Jussieu, 75005 Paris, France and Institut Universitaire de France
E-mail address, B. Després: `despres@ann.jussieu.fr`

Université Paris-Saclay, ENS Paris-Saclay, CNRS, Centre Borelli, F-91190 Gif-sur-Yvette, France
E-mail address, M. Ancellin: `matthieu.ancellin@ens-paris-saclay.fr`