A CONVERGENT DEEP LEARNING ALGORITHM FOR APPROXIMATION OF POLYNOMIALS

Bruno Després

A CONVERGENT DEEP LEARNING ALGORITHM FOR APPROXIMATION OF POLYNOMIALS

BRUNO DESPRÉS

Abstract. We start from the contractive functional equation proposed in [4], where it was shown that the polynomial solution of functional equation can be used to initialize a Neural Network structure, with a controlled accuracy. We propose a novel algorithm, where the functional equation is solved with a converging iterative algorithm which can be realized as a Machine Learning training method iteratively with respect to the number of layers. The proof of convergence is performed with respect to the L^∞ norm. Numerical tests illustrate the theory and show that stochastic gradient descent methods can be used with good accuracy for this problem.

1. Introduction

Neural Networks representations of real monovariate polynomials defined on the closed segment $x \in I = [0, 1]$ play a central role in the numerical analysis of Neural Networks (one can refer to [11, 13, 12, 4, 9, 2]). A central result is the Yarostky Theorem [15] which provides an approximation result of general functions, by means of a specific Neural Network approximation of the polynomial $x \mapsto x^2$ where the activation function is ReLU $R(x) = \max(0, x)$. This specific Neural Network can have an arbitrary large number of hidden layers, so it provides a simple example of a Deep Neural Network with perfectly known coefficients.

However, as pointed out by Ronald DeVore in 2019 [5], the stability of the approximation of polynomial functions by Deep Neural Networks (i.e. with many hidden layers) is not addressed in the current theory [7]. In particular it is already not the case for the Deep Neural Network which approximates the polynomial $x \mapsto x^2$. To the best of the understanding of the author of this Note, it is because all recent developments are devoted to abstract approximation theory and are scarcely related to constructive algorithms.

The present Note provides a positive answer to the Ronald DeVore's remark, by showing that there exists an algorithm with the following properties:
- the algorithm constructs a series of functions which intend to approximate given real polynomials in the reference segment I.
- under conditions, the algorithm is proved to be convergent in the L^∞ norm.
- it can be implemented in a Neural Networks/Machine Learning platform. The iterations of the algorithm directly corresponds to the number of hidden layers. The number of neurons which are trained per iteration of the algorithm is constant.
- numerical tests implemented in Tensorflow/Keras/Python [1] for very simple polynomials confirm the theoretical properties in terms convergence and stability.

Even if restricted to academic extremely simple functions, the Deep Learning algorithm presented in this Note seems the first one with a proof of convergence in L^∞ norm, where the iteration parameter is identical to the layer index. The reason is that the proposed formulation is not doomed with the curse of non convex optimization problems. Instead it solves a series of convex minimization problems. In the language of Neural Networks/Machine Learning, this Deep Network is trained

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

Keywords. Deep Learning, polynomials, Neural Networks, convergence.

by means of a series of Shallow Networks. The convergence is insured due to the underlying contractive structure explained in [4].

The Note is organized as follows. The setting of the problem is presented in Section 2. Discussion of a sharper contraction constant is the matter of Section 3. Section 4 is the core of the Note where we define a specific Machine Learning algorithm as a series of convex minimization problems. In Section 5, we show with an example that both contraction constants of Section 2 and 3 are non optimal. Final Section 6 is devoted to numerical tests which illustrate the theoretical properties. A discussion of the complexity of the algorithm and the presentation of some open problems are proposed at the end.

2. A functional equation

The notations and results are borrowed from a previous work [4]. They form the foundations on which the algorithm will be developed and justified.

The set of real polynomials is $P^n = \{p \text{ of degree } \leq n\}$. The set of continuous functions $C^0(I)$ over I is equipped with the maximal norm $\|f\|_{L^\infty(I)} = \max_{i\in I} |f(x)|$. We consider a subdivision in $m \geq 1$ subintervals $[x_j, x_{j+1}]$ where $0 = x_0 < x_1 < \cdots < x_j < \cdots < x_m = 1$ where $x_j = j\Delta x$ with $\Delta x = 1/m$. The set of continuous piecewise linear functions is

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

As in [4], we define the subset $E_h \subset V_h$

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

Functions in this set are called basis functions because of their central role, even if they are not classical Finite element basis functions. Finite Elements in V_h or in E_h are easy to implement with ReLU ($R(x) = \max(0, x)$) and with TReLU (ReLU with threshold) activation functions.
Once a real polynomial function $H \in P^n$ is given, the following problem is considered [4].

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*

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

with the contraction condition

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

If $n = 1$ the problem becomes trivial. That is why we only consider $n \geq 2$. The classical example [8, 15, 2] is $H(x) = x(1-x)$ which satisfies $H(x) = \frac{1}{4}g(x) + \frac{1}{4}H(g(x))$ where g is the normalized finite element function: $g(x) = 2x$ for $0 \leq x \leq \frac{1}{2}$ and $g(x) = 2(1-x)$ for $\frac{1}{2} \leq x \leq 1$. 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 $H(x) = e_0(x) + \frac{1}{4}H(e_1(x)) + \frac{1}{4}H(e_2(x))$ where $e_0(x) = \frac{1}{4}g(x)$. The contraction property (2.2) is satisfied with a constant $\sum |\beta_i| = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

Contrary to [4], we now completely specify the basis functions in order to optimize some approximations property and to explicit further implementation. We use a double index notation where $e_s = e_{j,k}$ for $1 \leq s \leq r$, $1 \leq j \leq m$ and $1 \leq k \leq n-1$ (necessarily $r = mn$). The basis functions are translated one from the other (this property was already stated in [4])

$$e_{j+1,k}(x) = e_{j,k}(x - \Delta x),$$

so the basis functions are completely determined by basis functions $e_{1,k}$. For $1 \leq k \leq n-1$ we take in this work

$$\begin{cases} e_{1,k}(x) = \frac{k-1}{2(n-1)}, & 0 \leq x, \\ e_{1,k}(x) = \frac{nm}{2(n-1)}x + \frac{k-1}{2(n-1)}, & 0 \leq x \leq 1/m \\ e_{1,k}(x) = \frac{n+k-1}{2km}, & 1/m \leq x. \end{cases} \tag{2.3}$$

Lemma 2.1. *For all possible indices $1 \leq j \leq m$ and $\leq 1 \leq k \leq n-1$, one has $e_{j,k} \in E_h$.*

Proof. Once the claim is proved for $e_{1,k}$, it will be generalized by translation to the other basis functions. By construction $e_{1,k}$ is continuous and piecewise affine. The left value is non negative $e_{1,k}(0) = \frac{k-1}{2(n-1)} \geq 0$. The right value is $e_{1,k}(1/m) = \frac{nm}{2(n-1)} \times \frac{1}{m} + \frac{k-1}{2(n-1)} = \frac{n+k-1}{2(n-1)} \leq \frac{n+n-2}{2(n-1)} \leq 1$. Therefore $x \mapsto e_{1,k}(x)$ takes its values between 0 and 1, which shows the claim. □

We remind the fundamental result proved recently in [4].

Theorem 2.1. *Let $H \in P^n$ with more precisely $deg(H) = n$. There exists a threshold value m_* such that the functional equation (2.1) has a solution with the contraction property (2.2) for all $m \geq m_*$.*

Proof. For sake of completeness, we provide a short proof adapted to the new notations.

• Consider the difference $q = H - \sum_{j=1}^{m}\sum_{k=1}^{n-1} \beta_{j,k} H \circ e_{j,k}$ where the coefficients are still unknowns at this stage. The second derivative of H is noted $p = H''$. In all intervals, one can calculate the second derivative $q'' \in P^{n-1}$. In the interval $I_j = [j/m, (j+1)/m]$ one has

$$q'' = p - \sum_{k=1}^{n-1} \beta_{j,k} \left(\frac{nm}{2(n-1)}\right)^2 p \circ e_{j,k}$$

The objective is to find coefficients $\beta_{j,k}$ such that $q'' = 0$ in I_j.

• The equality $q'' = 0$ in I_j is equivalent to $\frac{d^r}{dx^r} q''(x_j) = 0$ for $r = 0, \dots, n-2$, because q'' is a polynomial of degree $n-2$ in I_j. It yields a square linear system

$$\sum_{k=1}^{n-1} \left[\left(\frac{nm}{2(n-1)}\right)^{2+r} p^{(r)} \left(\frac{k-1}{2(n-1)}\right) \right] \beta_{j,k} = p^{(r)}(x_j), \qquad 0 \leq r \leq n-2.$$

The system is reorganized as

$$\sum_{k=1}^{n-1} p^{(r)} \left(\frac{k-1}{2(n-1)}\right) \beta_{j,k} = \left(\frac{2(n-1)}{nm}\right)^{2+r} p^{(r)}(x_j), \qquad 0 \leq r \leq n-2. \tag{2.4}$$

• The matrix of the linear system (2.4) is of Vandermonde type. Since the derivatives on the left hand side of (2.4) run for $r = 0$ to $r = n-1$, and the $n-1$ different evaluation points run for 0 to $\frac{1}{2}$, it is a classical matter that the matrix is invertible. Since the matrix does not depend on the interval index j, one obtains the bound

$$\max_k |\beta_{j,k}| \leq C/m^2 \text{ uniformly with respect to } j. \tag{2.5}$$

Therefore

$$K = \sum_{j=1}^{m}\sum_{k=1}^{n-1} |\beta_{j,k}| \leq C(n-1)/m, \tag{2.6}$$

which yields the contraction property for $m \geq m^*$ high enough.

• With these coefficients, the difference q is a continuous function with vanishing second derivatives in all I_j. Therefore one can write $q = -e_0 \in V_h$ and the claim is proved. □

What distinguishes the family (2.3) with the more general family considered in [4][Theorem 1] is that the slope of the basis functions is always > 1. Indeed the slope $\frac{nm}{2(n-1)}$ is by construction the same for all basis functions (2.3), so since $m \geq 2$ the property is evident. Considering (2.4), the greater the slope the smaller the right hand side of the linear system. This is a reason why it is possible to think that the family (2.3) provides better control of the coefficients $\beta_{j,k}$, so ultimately is optimal to obtain a smaller contraction constant K, see (2.6).

3. A sharper contraction constant

The right hand side operator (2.1) written for the new basis functions is

$$\left\{ \begin{array}{llll} \mathcal{H}: & L^\infty(I) & \to & L^\infty(I) \\ & G & \mapsto & \sum_{j=1}^{m} \sum_{k=1}^{n-1} \beta_{j,k} G \circ e_{j,k}. \end{array} \right. \tag{3.1}$$

It is endowed with the contraction property (2.2)

$$\|\mathcal{H}G\|_{L^\infty(I)} \leq K \|G\|_{L^\infty(I)}, \qquad K = \sum_{j=1}^{m} \sum_{k=1}^{n-1} |\beta_{j,k}| < 1. \tag{3.2}$$

Such property is central for the justification of the Deep Learning algorithm of this Note.

The purpose in this Section is to present a sharper bound. Let us define

$$\widetilde{K} = 2 \left(\max_{j=1}^{m} \sum_{k=1}^{n-1} |\beta_{j,k} \right).$$

Lemma 3.1. *One has the general bound:* $\inf_{e \in V_h} \|\mathcal{H}(G) - e\|_{L^\infty(I)} \leq \widetilde{K} \|G\|_{L^\infty(I)}$.

Remark 3.1. The new contraction constant is sharper asymptotically for large m, because the sum with respect to m in (3.2) is removed. Using (2.5), one gets the bound $\widehat{K} \leq C(n-1)/m^2$ which is better than (2.6).

Proof. We write $\mathbf{1}_j$ the indicatrix function of the interval I_j, that is $\mathbf{1}_j(x) = 1$ for $(j-1)/m < x < j/m$ and $\mathbf{1}_j(x) = 0$ for $x < (j-1)/m$ or $j/m < x$.
Let $x_{j+\frac{1}{2}} = (j + \frac{1}{2})/m$ be the central point in the interval I_j. One can write

$$\mathcal{H}(G) = \sum_j \sum_k \beta_{j,k} G \circ e_{j,k} \times \mathbf{1}_j + \sum_j C_j \mathbf{1}_j$$

where

$$C_j = \sum_{i<j} \sum_k \beta_{i,k} G \circ e_{i,k}(x_{j+\frac{1}{2}}) + \sum_{j<i} \sum_k \beta_{i,k} G \circ e_{i,k}(x_{j+\frac{1}{2}}). \tag{3.3}$$

Let $e \in V_h$. A triangular inequality yields that

$$\|\mathcal{H}(G) - e\|_{L^\infty(I)} \leq \left\| \sum_j \sum_k \beta_{j,k} G \circ e_{j,k} \times \mathbf{1}_j \right\|_{L^\infty(I)} + \left\| \sum_j C_j \mathbf{1}_j - e \right\|_{L^\infty(I)}. \tag{3.4}$$

The first term is bounded as

$$\left\| \sum_j \sum_k \beta_{j,k} G \circ e_{j,k} \times \mathbf{1}_j \right\|_{L^\infty(I)} \leq \left(\max_{j=1}^{m} \sum_{k=1}^{n-1} |\beta_{j,k} \right) \|G\|_{L^\infty(I)}. \tag{3.5}$$

To bound the second term, we design a piecewise affine function $e \in V_h$ as follows

$$e(x_0) = C_1, \quad e(x_j) = \frac{C_j + C_{j+1}}{2} \text{ for } 1 \leq j \leq m_1, \quad e(x_m) = C_m.$$

A proof by drawing shows that

$$\left\|\sum C_j \mathbf{1}_j - e\right\|_{L^\infty(I)} \leq \max_{j=1}^{m-1} \frac{|C_{j+1}-C_j|}{2}. \tag{3.6}$$

The definition of the constants C_j and C_{j+1} yields that

$$\begin{array}{rcl} C_{j+1}-C_j & = & \sum_{i<j}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{3}{2}}) + \sum_k \beta_{j,k} G\circ e_{j,k}(x_{j+\frac{3}{2}}) + \sum_{j+1<i}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{3}{2}}) \\ & - & \sum_{i<j}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{1}{2}}) - \sum_k \beta_{j+1,k} G\circ e_{j+1,k}(x_{j+\frac{1}{2}}) - \sum_{j<i}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{1}{2}}). \end{array}$$

By definition (2.3), the basis functions are constant on the left and on the right of the interval where they vary linearly. Therefore

$$\sum_{i<j}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{3}{2}}) - \sum_{i<j}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{1}{2}}) = 0$$

and

$$\sum_{j+1<i}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{3}{2}}) - \sum_{j<i}\sum_k \beta_{i,k} G\circ e_{i,k}(x_{j+\frac{1}{2}}) = 0.$$

So one deduces that

$$\begin{array}{rcccc} |C_{j+1}-C_j| & = & \left|\sum_k \beta_{j,k} G\circ e_{j,k}(x_{j+\frac{3}{2}})\right. & - & \left.\sum_k \beta_{j+1,k} G\circ e_{j+1,k}(x_{j+\frac{1}{2}})\right| \\ & \leq & \sum_k |\beta_{j,k}| \|G\|_{L^\infty(I)} & + & \sum_k |\beta_{j+1,k}| \|G\|_{L^\infty(I)} \end{array} \tag{3.7}$$

which turns into

$$|C_{j+1}-C_j| \leq 2\left(\max_{j=1}^{m}\sum_{k=1}^{n-1}|\beta_{j,k}\right)\|G\|_{L^\infty(I)}. \tag{3.8}$$

The claim is a consequence of (3.4-3.8). □

4. A Deep Machine Learning algorithm

This Section comes to the core of this Note since by presenting an abstract Deep Machine Learning algorithm endowed with a proof of convergence with respect to the number of layers (so the name Deep Learning). This Deep Learning algorithm can be implemented in one of the Neural Networks/Machine Learning softwares freely available such as Tensorflow/Keras [1], Scikit-Learn (Inria[1]), Pytorch (Facebook[2]), Julia (MIT licence[3]). This list is not exhaustive.

It is sufficient for this presentation to consider that Neural Networks/Machine Learning softwares have two features: a) they manipulate functions assembled with ReLU like activation functions and composition of functions, b) they perform numerical optimisation to fit the coefficients, in particular with stochastic gradient descent methods [7, 3].

We assume that a certain number of basis functions $e_1, \ldots, e_r$ are decided. The basis functions are assembled with the ReLU activation functions and alike. In the context of this work, a natural choice is to take the basis functions defined by (2.3). We will write $\beta = (\beta_1, \ldots, \beta_r) \in \mathbb{R}^r$. For $(e_0, \beta, f) \in V_h \times \mathbb{R}^r \times L^\infty(I)$, we will write $g(e_0, \beta, f) \in L^\infty(I)$ the function defined by

$$g(e_0,\beta,f) = e_0 + \sum_{i=1}^r \beta_i f\circ e_i.$$

[1] `https://scikit-learn.org/stable/`

[2] `https://pytorch.org`

[3] `https://julialang.org`

Let us decide of a given polynomial $H \in P^n$. One constructs a cost function where the main variables to optimize are e_0 and β and the parameters are the functions f and H

$$J(e_0, \beta : f, H) = \|g(e_0, \beta, f) - H\|_{L^\infty(I)}$$

The algorithm below defines a sequence of functions $(f_k)_{k\in\mathbb{N}}$ as the solution of minimization problems.

- **Initialization:** The seed is the null function
$$f_0(x) = 0 \text{ for all } x \in I. \tag{4.1}$$
- **Iterations on k:** The next function is $f_{k+1} = g(e_0, \beta, f_k)$ where (e_0^k, β^k) is any solution of the minimization problem
$$(e_0^k, \beta^k) = \underset{(d_0,\alpha)\in V_h\times\mathbb{R}^r}{\arg\min} \|g(d_0, \alpha, f_k) - H\|_{L^\infty(I)}. \tag{4.2}$$

After comments on the Neural Network implementation of (4.1-4.2) , we will show the convergence $f_k \to H$ in the L^∞ norm.

Lemma 4.1. *The number of hidden layers in a Neural Network implementation of f_k is $k-1$.*

Proof. Since $f_0 = 0$ then $f_1 = g(e_0^1, \beta^1, 0) \in V_h$. Then $f_2 = g(e_0^2, \beta^2, f_1)$ can be implemented in a Neural Network with one hidden layer. Then $f_3 = g(e_0^3, \beta^3, f_2)$ can be implemented in a Neural Network with two hidden layers, and so one and so forth. □

The algorithm is correctly defined as shown by the next result.

Lemma 4.2. *The minimization problem is convex (but not strictly convex), and has always at least one solution. For any $(f, H) \in L^\infty(I) \times L^\infty(I)$, there exists a minimizer $(e_0, \beta) \in V_h \times \mathbb{R}^r$ such that*

$$J(e_0, \beta : f, H) \le J(d_0, \alpha : f, H) \qquad \text{for all } (d_0, \alpha) \in V_h \times \mathbb{R}^r.$$

The minimizer (e_0, β) is a priori non unique.

Proof. Such a proof is standard in convex analysis in finite dimension [10].
The dimension of the linear space V_h is equal to $m+1$, so any function $h_0 \in V_h$ admits the linear expansion $h_0 = \sum_{p=1}^{m+1} \delta_p \varphi_p$ where the coefficients of the linear expansion are $\delta_p \in \mathbb{R}$ and $V_h = \text{Span}(\varphi_p)_{1\le p\le m+1}$. Then one can write

$$g(h_0, \gamma, f) = \sum_{p=1}^{m+1} \delta_p \varphi_p + \sum_{i=1}^{r} \beta_i f \circ e_i \tag{4.3}$$

which shows that $g(h_0, \gamma, f)$ belongs to the linear subspace of $L^\infty(I)$ generated by the $(\varphi_p)_p$ and the $(f \circ e_i)_i$. Let us take a linear basis in the vectorial space, the basis has a dimension $m+1 \le q \le m+1+r$, so the sum can be recovered in function on q coefficients only. It is a classical exercice (not reproduced here) to show that the cost function $J(f_0, \gamma : f, H)$ is coercive with respect to these q coefficients: that is the cost function tends to $+\infty$ as a norm of these q coefficients tends to $+\infty$. Since the cost function is bounded from below, then it has a minimum. Finally it is a standard matter that the L^∞ norm does not bring strict coercivity so uniqueness is not guaranteed. □

As a consequence the sequence defined by the sequence of minimization problems (4.2) is a priori non unique. Nevertheless it is convergent.

Theorem 4.1. *Assume (2.1-2.2). Then a series (4.1-4.2) converges with the bound*

$$\|f_k - H\|_{L^\infty(I)} \le \widehat{K}^k \|H\|_{L^\infty(I)}, \qquad \widehat{K} = \min(K, \widetilde{K}).$$

Proof. By (4.2) one has the bound $\|f_{k+1}-H\|_{L^\infty(I)} \le \|d_0 + \sum_{i=1}^r \alpha_i f_k \circ e_i - H\|_{L^\infty(I)}$ for all $d_0 \in V_h$ and all $\alpha = (\alpha_1, \dots, \alpha_r) \in \mathbb{R}^r$.
• Let us take $(d_0, \alpha) = (e_0, \beta)$ in accordance with the representation (2.1-2.2). One deduces the inequality

$$\begin{array}{rcl} \|f_{k+1} - H\|_{L^\infty(I)} & \le & \|e_0 + \sum_{i=1}^r \beta_i f_k \circ e_i - e_0 - \sum_{i=1}^r \beta_i H \circ e_i\|_{L^\infty(I)} \\ & \le & \|\sum_{i=1}^r \beta_i (f_k - H) \circ e_i\|_{L^\infty(I)} \\ & \le & \sum_{i=1}^r |\beta_i| \|f_k - H\|_{L^\infty(I)} \\ & \le & K \|f_k - H\|_{L^\infty(I)}. \end{array}$$

• One can also take $(d_0, \alpha) = (e_0 - e, \beta)$ where $e \in V_h$ is arbitrary. One obtains

$$\begin{array}{rcl} \|f_{k+1} - H\|_{L^\infty(I)} & \le & \|e_0 - e + \sum_{i=1}^r \beta_i f_k \circ e_i - e_0 - \sum_{i=1}^r \beta_i H \circ e_i\|_{L^\infty(I)} \\ & \le & \|\mathcal{H}(f_k - H) - e\|_{L^\infty(I)} \\ & \le & \widetilde{K} \|f_k - H\|_{L^\infty(I)} \end{array}$$

by virtue of (3.2) and Lemma 3.1.
• So $\|f_{k+1} - H\|_{L^\infty(I)} \le \widehat{K} \|f_k - H\|_{L^\infty(I)}$. Since $f_0 = 0$, the claim is obtained by iteration on k. □

5. Non optimality of $\widehat{K}$

An elementary exemple that will be used for the numerical tests is the following. For this example one can check that $\widehat{K} > 1$. However we will see that a sharper value constant exists at the end of the Section.

Lemma 5.1. *Take the third order Tchebycheff polynomial rescaled in* $[0, 1]$*, that is* $H(x) = T_3(2x - 1) = 32x^3 - 48x^2 + 18x - 1$. *Take* $m = 3$ *and* $n = 3$.
Then there exists $e_0 \in V_h$ *such that* $H = e_0 + \sum_{j=1}^3 \sum_{k=1}^2 \beta_{j,k} H \circ e_{j,k}$ *with*

$$\beta_{1,1} = \beta_{3,2} = \frac{2^5}{3^4} - \frac{2^6}{3^6}, \quad \beta_{1,2} = \beta_{3,1} = \frac{2^7}{3^6} - \frac{2^5}{3^4}, \quad \beta_{2,1} = \beta_{2,2} = \frac{2^5}{3^6}.$$

Proof. The difference $q = H - \sum_{j=1}^2 \sum_{k=1}^2 \beta_{j,k} H \circ e_{j,k}$ is a continuous function. It is a polynomial of degree at most 3 in the three intervals $I_1 = [0, 1/3]$, $I_2 = [1/3, 2/3]$ and $I_3 = [2/3, 1]$.
• In I_1 one has

$$e_{1,1}(x) = \frac{9}{4}x, \quad e_{1,2}(x) = \frac{9}{4}x + \frac{1}{4}, \quad e_{2,1},\ e_{2,2},\ e_{3,1} \text{ and } e_{3,2} \text{ are constant.}$$

Then $H(e_{1,1}(x)) = 32\left(\frac{9}{4}x\right)^3 - 48\left(\frac{9}{4}x\right)^2 + h_1 x + h_2$ and $H(e_{1,2}(x)) = 32\left(\frac{9}{4}x\right)^3 - 24\left(\frac{9}{4}x\right)^2 + h_3 x + h_4$. Making vanish in I_1 the coefficients of the third order and second order monomials in q yields the system

$$\left\{ \begin{array}{rcl} \beta_{1,1} + \beta_{1,2} & = & \left(\frac{4}{9}\right)^3 = \frac{2^6}{3^6}, \\ 2\beta_{1,1} + \beta_{1,2} & = & 2\left(\frac{4}{9}\right)^2 = \frac{2^5}{3^4}. \end{array} \right.$$

The solution is $\beta_{1,1} = \frac{2^5}{3^4} - \frac{2^6}{3^6}$ and $\beta_{1,2} = \frac{2^7}{3^6} - \frac{2^5}{3^4}$.
• Concerning I_3 one notices that the third order Tchebycheff is antisymmetric with respect to the center $x = 1/2$. It explains why $\beta_{1,1} = \beta_{3,2}$ and $\beta_{1,2} = \beta_{3,1}$. These values can also be obtained after lengthy but elementary calculations.
• In I_2 one has

$$e_{2,1}(x) = \frac{9}{4}x - \frac{3}{4}, \quad e_{2,2}(x) = \frac{9}{4}x - \frac{1}{2}, \quad e_{1,1},\ e_{1,2},\ e_{3,1} \text{ and } e_{3,2} \text{ are constant.}$$

One checks that $H(e_{2,1}(x)) = 32\left(\frac{9}{4}x\right)^3 - 120\left(\frac{9}{4}x\right)^2 + h_5x + h_6$ and $H(e_{2,2}(x)) = 32\left(\frac{9}{4}x\right)^3 - 96\left(\frac{9}{4}x\right)^2 + h_7x + h_8$. The linear system which corresponds to the annulation in q of the monomials x^3 and x^2 can be written

$$\left\{\begin{array}{lll} \beta_{2,1} + \beta_{2,2} & = & \left(\frac{4}{9}\right)^3 = \frac{2^6}{3^6}, \\ 120\beta_{2,1} + \beta_{2,2} & = & 48\left(\frac{4}{9}\right)^2. \end{array}\right.$$

The solution is $\beta_{2,1} = \beta_{2,2} = \frac{2^5}{3^6}$.

- Therefore q is continuous and piecewise affine. One can write $q = -e_0 \in V_h$. □

Remark 5.1. A numerical application shows that $\beta_{1,1} = \beta_{3,2} = 0.3072\ldots$, $\beta_{1,2} = \beta_{3,1} = -0.2194\ldots$ and $\beta_{2,1} = \beta_{2,2} = 0.0438\ldots$ With these values $K > \widetilde{K} > 1$. However the passage in the proof from estimate (3.7) to estimate (3.8) is non optimal because it looses the fact that (3.7) concerns two consecutive indices. For $m = 3$, two consecutive indices can be either $(j, j+1) = (1, 2)$ or $(j, j+1) = (2, 3)$. Then, for the third order rescaled Tchebycheff polynomial, one gets the sharper bound

$$\overline{K} = |\beta_{1,1}| + |\beta_{1,2}| \frac{1}{2}\left(|\beta_{1,1}| + |\beta_{1,2}| + |\beta_{2,1}| + |\beta_{2,2}|\right) = \frac{3}{2}\left(|\beta_{1,1}| + |\beta_{1,2}|\right) + |\beta_{2,1}| = 0.8340\ldots \tag{5.1}$$

Now $\overline{K} < 1$. This bound will be invoked to justify the numerical convergence in test #2.

6. Numerical tests and discussion

Algorithm (4.1-4.2) has been implemented in Tensorflow/Keras [1] for the purpose of numerical illustrations. The softwares `Branch_data.py` (training) and `Branch_gene.py` (data generation) written for the tests are available at the Git repository `https://github.com/despresbr/NNNA`. The minimisation problems (4.2) are performed with the ADAM algorithm which is a stochastic descent gradient method with batches. A dataset is created with oversampling and the cost function in the $L^\infty(I)$ norm is defined by creating a user-made loss function. The function $g(e_0, \beta, f_k)$ is assembled by using the concatenation-of-layers technique [1][page 243]. The CPU cost of the learning stage is the same for all iterations k since the number of free parameters (i.e. the number of neurons in this case) is the same at each stage of the algorithm because we ask for the same number of epochs and the same size of the batches. We use a trick which is classical for algorithms which have inner loops (new training) inside a global exterior loop (iteration on k). That is the starting point for a new training is the end point of the previous training. It helps to save computational efforts. We also follow the prescription [14, 3] where a decrease of the gradient length (i.e. the learning rate in Machine Learning language) is advocated.

The minimization in the L^∞ norm is performed approximatively because it is not the objective of stochastic gradient descent algorithms to calculate global minima with sharp accuracy. In particular we do not know precisely the influence of the batches on the accuracy of the minimization procedure. Despite this fact, we observe strong convergence in the tests below where the error $\epsilon_k = \|f_k - H\|_{L^\infty(I)}$ is reported in function of the iteration index k. The tests below have been calculated on a GPU node at SCAI-Sorbonne university[4]. They show convergence of algorithm (4.1-4.2) in accordance with the main Theorem of convergence 4.1.

[4] https://scai.sorbonne-universite.fr

6.1. **Test #1.** The polynomial is $H(x) = x - x^2$. The training dataset is made with 4000 pairs $(x_s, y_s = H(x_s))_{1 \leq s \leq 4000}$ where x_s is sampled uniformly between 0 and 1. Around 20% of the pairs are taken outside for validation. The batches are made with 128 pairs. The error at iteration k is $\epsilon_0 = 0.0377$, $\epsilon_1 = 0.00806$, $\epsilon_2 = 0.00238$, $\epsilon_3 = 0.000521$, $\epsilon_4 = 0.000165$, $\epsilon_5 = 3.66e - 5$ then $\epsilon_6 = 9.97e - 6 - 5$. The numerical rate of convergence, that is the numerical contraction constant, is $K_{\text{num}} \approx \frac{1}{4}$ which is better than the theoretical one.

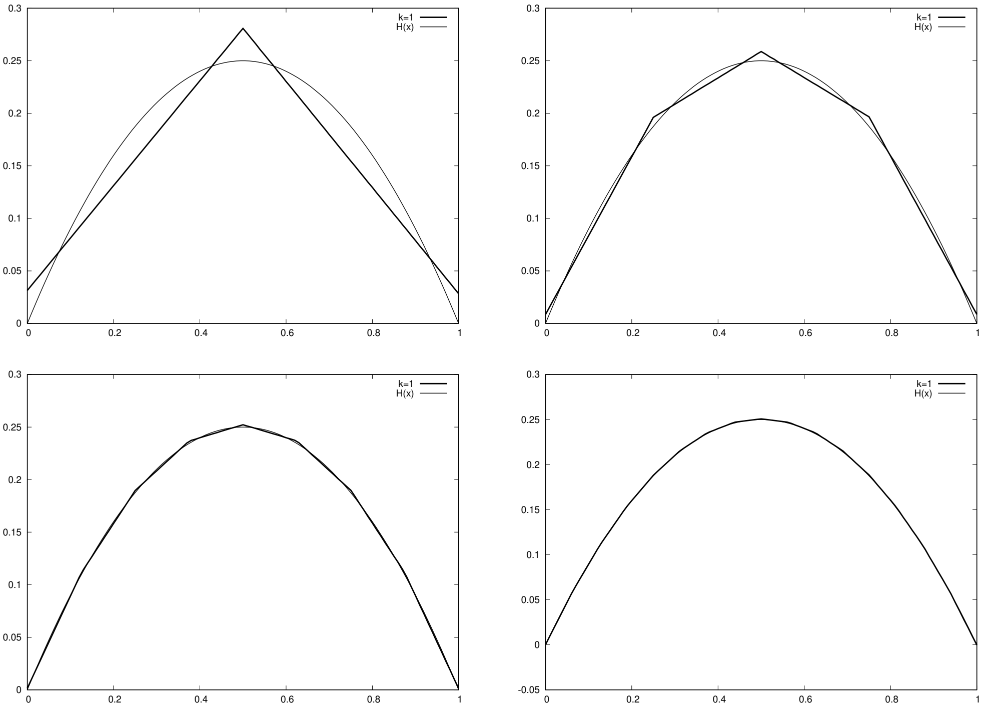

FIGURE 6.1. Display of the function calculated by the algorithm versus the objective H. The four plots corresponds to $0 \leq k \leq 3$. One observes numerical convergence.

6.2. **Test #2.** The polynomial is the rescaled Tchebycheff polynomial of Lemma 5.1: $H(x) = 32(2x-1)^3 - 48(2x-1)^2 + 18x - 1$. Theoretical convergence is guaranteed by means of the estimate of Remark 5.1. We take the same parameters as in the previous test except that $m = 3$ and $n = 2$. The error at iteration k is $\epsilon_0 = 0.459$, $\epsilon_1 = 0.179$, $\epsilon_2 = 0.0834$, $\epsilon_3 = 0.0409$, $\epsilon_4 = 0.0109$, then $\epsilon_5 = 0.00471$. The numerical convergence is observed. It is compatible with the fact that $\overline{K} < 1$ (see (5.1)). However $\overline{K}$ is quite close to 1, so it cannot explain the observed fast convergence with a factor $\approx 1/2$.

6.3. **Test #3.** We perform the same test as in the previous one, except that now $m = 5$ and $n = 3$. We report only the accuracy of the first four iterations of the algorithm, because the computational cost related to the manipulation of functions increases for $k = 5$. The accuracy is $\epsilon_0 = 0.201$, $\epsilon_1 = 0.027$, $\epsilon_2 = 0.0057$ and $\epsilon_3 = 0.00103$. One observes a more pronounced rate of convergence, in accordance with bound (2.6) and Lemma 3.1.

6.4. **Test #4.** Finally we consider the fifth order rescaled Tchebycheff polynomial: $H(x) = 16(2x-1)^5 - 20(2x-1)^3 + 5(2x-1)$, and we take $m = 9$ and $n = 5$. The errors are $\epsilon_0 = 0.379$, $\epsilon_1 = 0.0625$, $\epsilon_2 = 0.0318$ and $\epsilon_3 = 0.00947$. A comparison of the exact solution and the numerical solution at step $k = 3$ is proposed in Figure 6.2.

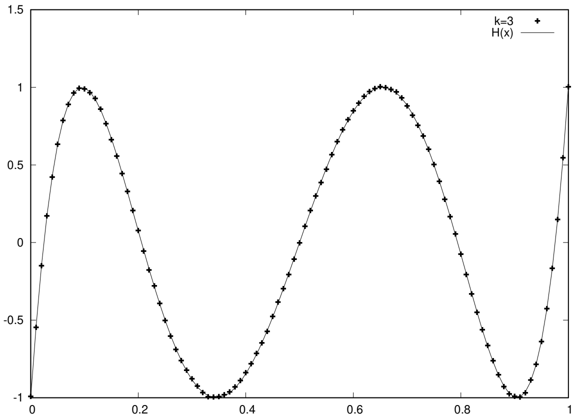

FIGURE 6.2. Fifth order rescaled Tchebycheff polynomial: numerical solution (cross) versus exact function (solid line).

6.5. **Discussion.** We discuss the complexity of the method and present three open problems.

6.5.1. *Complexity.* The complexity of the algorithm can be defined as the relation between the accuracy and the cost in the asymptotic range $k >> 1$. We present hereafter simple bounds.

• As a consequence of Theorem 4.1, the accuracy in L^∞ norm naturally scales like

$$\varepsilon = O(K^k)$$

where we remind that k is the number of layers and $K < 1$ is the contraction constant.

• The cost can be estimated in two ways. Either the cost is estimated as the number of neurons to train, equal to the number of free parameters in (4.3) times the number of layers. Since $r = m(n-1)$, one obtains $C = O(mnk)$. It yields the complexity scaling

$$C = O\left(|\log \varepsilon|\right) \tag{6.1}$$

which is similar to the one of the Yarotsky Theorem [15].

• Or the cost is estimated is the number of calculations to perform to calculate one function f_k. Due to the hierarchical structure of the whole method, it scales like $C = O((mn)^k)$. It yields the complexity scaling

$$C = O\left(\frac{1}{\varepsilon^{\frac{\log mn}{|\log K|}}}\right) \tag{6.2}$$

which is of course much worse than (6.1). For a practical calculation, the relation cost/accuracy is a compromise between these two bounds, depending on the parts which are the most costly. However the numerical experiments clearly show that the scaling (6.2) is the issue.

• A related problem is that the scaling $(mn)^k$ may induce an important CPU cost, independently of the level of accuracy. This is related to the technology for hierarchical declaration of functions used in Machine Learning softwares. It is possible that this computational cost is lessen by using a different implementation, but it is not clear and so far this is a huge concern. We have observed that a way to get a control on it and to obtain reasonable accuracy at reasonable cost is to take m large and $k \leq 3$ (as in test #4).

6.5.2. *Three open problems.* An open mathematical problem is to understand how to recover an optimal contraction/approximation constant in maximal norm, using perhaps technics from Constructive Approximation theory [6].

An open numerical problem left for further research is the optimization basis functions.

A more general problem is to determine if there could be a way to extrapolate the approach used in this Note for the approximation of more general non polynomial functions in higher dimensions. Any progress in this direction would be of critical importance for the numerical analysis of Machine Learning techniques applied to real life problems. So far, it is a completely open axis of research.

References

[1] F. Chollet, *Deep Learning with Python*, Manning Shelter Island, 2018.

[2] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, G. Petrova, "Nonlinear Approximation and (Deep) ReLU Networks", *Constructive Approximation* **55/1** (2022), p. 127-172.

[3] B. Després, *Neural Networks and Numerical Analysis*, De Gruyter, 2022.

[4] B. Després, M. Ancellin, "A functional equation with polynomial solutions and application to Neural Networks", *Comptes Rendus Mathématique* **358** (2020), no. 9-10, p. 1059-1072.

[5] R. DeVore, "Nonlinear Approximation by Deep ReLU Neural Networks", `https://devore2019.sciencesconf.org/resource/page/id/1`, 2019.

[6] R. A. DeVore, G. G. Lorenz, *Constructive Approximation*, Springer, 1993.

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

[8] M. Hata, M. Yamaguti, "The Takagi Function and Its Generalization", *Japan J. AppL Math.* **1** (1984), p. 83-199.

[9] J. He, L. Li, J. Xu, C. Zheng, "ReLU Deep Neural Networks and Linear Finite Elements", *Journal of Computational Mathematics* **38** (2020), no. 3, p. 502-527.

[10] J. B. Hiriart-Urruty, C. Lemaréchal, *Convex analysis and minimization algorithms. I*, vol. 305, Springer-Verlag, Berlin, 1993.

[11] Y. Le Cun, *Quand la machine apprend*, Odile Jacob, 2019, in French.

[12] B. Li, S. Tang, H. Yu, "Better approximations of high dimensional smooth functions by deep neural networks with rectified power units", *Commun. Comput. Phys.* **27** (2020), no. 2, p. 379-411.

[13] J. A. A. Opschoor, P. C. Petersen, C. Schwab, "Deep ReLU networks and high-order finite element methods", *Anal. Appl. (Singap.)* **18** (2020), no. 5, p. 715-770.

[14] G. Turinici, "The convergence of the Stochastic Gradient Descent (SGD) : a self-contained proof", arxiv arXiv:2103.14350, 2021.

[15] 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

E-mail address, B. Després: `bruno.espres@sorbonne-universite.fr`