Adaptive numerical approximation and neural networks

Bruno Després

January 10, 2021

1 Introduction

On the one hand, Artificial Intelligence, Deep Learning, Neural Networks, ..., are names in the air showing great promises. On the other hand, for many problems in applied science and industry, the question is to model a given **non-linear** phenomenon or function $f : \mathbb{R}^m \to \mathbb{R}^n$ with an input-output approach and **guaranteed accuracy**. It is precisely this class of problems which is the target of this course.

Motivated by these issues, this course will adopt the viewpoint that adaptive interpolation or **adaptive numerical approximation** with prescribed or **controlled accuracy** of functions known by their values a given points is the target. These adaptive approximations are constructed within a (deep) **neural network**. The errors are measured in various discrete L^p or H^1 norms, or in the sense of the entropy for logistic regression. We will review the basic concepts of adaptive numerical approximation with a neural network. Our objective is to present in a unified framework some basic mathematical and algorithmic principles at the foundations of numerical approximations with neural networks, to introduce some modern softwares and techniques in relation with these principles and to present the application to the numerical discretization of simple partial differential equations. The emphasis will be:

- On practical calculation of the weights and the approximation of the objective function with methods with controlled accuracy.
- On the structure of specific neural networks will be described in detail for the Takagi function, the calculation of H^1 norm, ... The connection with the theory of Finite Element discretization will be presented for the case where the activation function is the rectifie linear unit function (ReLU).
- On the interpretation of gradient descent algorithms and techniques by means of ordinary differential equations (EDO).
- On the application to the numerical approximation of partial differential equations (EDP) such as the advection equation
$$\partial_t \alpha + \mathbf{a} \cdot \nabla \alpha = 0$$
and various other problems.

The current limitations will be highlighted. Practical numerical approximation problems coming from the **numerical modeling of fluids** will serve either as illustrations/complements and research projects on the computer will be encouraged.

References

[1] F. Cholet, Deep-Learning with Python, Manning, 2018.

[2] B. Després and H. Jourdren, Machine Learning design of Volume of Fluid schemes for compressible flows, JCP 2020.

[3] F. Gibou, D. Hyde and R. Fedkiw, Sharp interface approaches and deep learning techniques for multiphase flows, JCP 2019.

[4] I. Goodfellow, Y. Benjio, A. Courville, Deep Learning, 2015.

[5] D. Ray and J. Hesthaven, An artificial neural network as a troubled-cell indicator, JCP 2018.

Contents

2 Objective functions and neural networks

The basic problems of the notes are formulated in the context of interpolation of functions. Let f^{obj} be an objective function which models some problem of interest

$$f^{\text{obj}} : \mathbb{R}^m \longrightarrow \mathbb{R}^n. \tag{1}$$

Let x_i be an interpolation point in $\mathbb{R}^m$. Let $\varepsilon_i \in \mathbb{R}^n$ which denotes a noise (small enough in a sense which needs to be specified). Noise is an important notion for real data, because real measurements or real data are never perfect. Set

$$y_i = f^{\text{obj}}(x_i) + \varepsilon_i \in \mathbb{R}^n. \tag{2}$$

The pair $(x_i, y_i) \in \mathbb{R}^m \times \mathbb{R}^n$ is called an interpolated data with noise $\varepsilon_i \in \mathbb{R}^m$. If $\varepsilon_i = 0$, the interpolation data is noiseless. The necessarily finite collection of interpolation data will be called the dataset

$$\mathcal{D} = \{(x_i, y_i),\ i = 1, \dots\} \subset \mathbb{R}^m \times \mathbb{R}^n. \tag{3}$$

Let us note in passing that it is clearly important to have good datasets where the pairs (x_i, y_i) sample the objective function in a region which is meaningful.

A general question is to implement in a computer a good approximation denoted as f of the objective function f^{obj}, starting from a given dataset $\mathcal{D}$. Some evident difficulties for the design of f are that the level of the noise, the curse of dimension (that is m and/or n take high values), the construction of good datasets, the fact that the implementation must be efficient for large I, the quality of the reconstructed function. Other aspects will be developed in the notes.

2.1 Least squares

A first ingredient is the least squares method[1]. Consider a linear function f

$$\begin{array}{rccl} f: & \mathbb{R}^m & \longrightarrow & \mathbb{R}^n, \\ & x & \longmapsto & f(x) = Wx + b \end{array} \tag{5}$$

where the parameters are $W \in \mathcal{M}_{mn}(\mathbb{R})$ which is called the the matrix of weights and $b \in \mathbb{R}^n$ which is called the offset. Consider now the function

$$\begin{array}{rccl} J: & \mathcal{M}_{mn}(\mathbb{R}) \times \mathbb{R}^n & \longrightarrow & \mathbb{R}, \\ & (W, b) & \longmapsto & \displaystyle\sum_{(x,y)\in\mathcal{D}} |Wx + b - y|^2 . \end{array} \tag{6}$$

[1]Standard notations are as follows. For an arbitrary vector $z \in \mathbb{R}^p$ in an arbitrary dimension p, we note its square $|z|^2 = \sum_{i=1}^p |z_i|^2 \geqslant 0$. One has $|z|^2 = \langle z, z\rangle$ where the the scalar product $\langle \cdot, \cdot \rangle$ is defined by

$$\langle a, b\rangle = \sum_{i=1}^p a_i b_i, \qquad a, b \in \mathbb{R}^p.$$

For two vertical vectors $a, b \in \mathbb{R}^m$ such that $b = Mc$ with $c \in \mathbb{R}^n$ and the matrix is $M \in \mathbb{M}_{mn}$, one has

$$\langle a, b\rangle = \langle a, Mc\rangle = \langle a \otimes c : M\rangle$$

where : denotes the contraction of matrices of equal size and $\otimes$ denotes the tensorial product of two vectors $a \otimes c = ac^t \in \mathcal{M}_{mn}(\mathbb{R})$. A norm for matrices writes

$$\|W\| = \sup_{\|y\|=1, \|x\|=1} \langle y, Mx\rangle = \sup_{\|y\|=1, \|x\|=1} \langle y \otimes x : M\rangle . \tag{4}$$

The space of parameters is noted $\mathcal{Q} = \mathcal{M}_{nm}(\mathbb{R}) \times \mathbb{R}^n = \mathcal{M}_{n,m+1}(\mathbb{R})$. A way to minimize the difference between f^{obj} which is encoded in the dataset $\mathcal{D}$ and the function f is to determine the parameters which minimize J. That is we consider (or hope) that an optimal value (in the sense of least squares) of the parameters $(W_*, b_*) \in Q$ satisfies

$$J(W_*, b_*) \leqslant J(W, b) \qquad \forall (W, b) \in Q.$$

Lemma 2.1. *There exists an optimal value* $(W_*, b_*) \in Q$

$$(W_*, b_*) = \underset{(W,b)\in Q}{\operatorname{argmin}}\ J(W, b).$$

If the vectors x_i *span the whole space* $\mathbb{R}^n$*, that is* $Span(x_i)_{i=1}^{\#\mathcal{D}} = \mathbb{R}^m$*, then the optimal value is unique.*

Remark 2.2. *For most problems in datascience* $\#\mathcal{D} >> 1$*, so the assumption is a reasonable one.*

Proof. The optimal matrix of weights and offset are constructed explicitly. The proof is split in smaller steps.

- In the first step we freeze $b = 0$ and look for the optimal value of W.
- In the second step b and W can take any value.
- For these two steps, we make an additional hypothesis about the linear independence of the vectors x_i because it simplifies the proofs. This hypothesis of linear independence of the vectors x_i is relaxed in the final comment.

First step. Consider the function $J(W) = \frac{1}{2}\sum_{i=1}^{N} |Wx_i - y_i|^2$ where the factor $\frac{1}{2}$ is just for convenience. Set $h(\varepsilon) = J(W + \varepsilon Z)$ where $Z \neq 0$ is an arbitrary non zero matrix $Z \in \mathcal{M}_{mn}(\mathbb{R})$. At a local minimum W_*, the derivative of h vanishes whatever Z is . One has

$$\lim_{\varepsilon} \frac{h(\varepsilon) - h(0)}{\varepsilon} = \sum_{i=1}^{N} \langle W_* x_i - y_i, Z x_i \rangle = 0,$$

The scalar product can be rewritten as a matrix-matrix or vector-vector multiplication as

$$\sum_{i=1}^{N} (Wx_i - y_i)^t \, Z x_i = 0.$$

One more transposition yields

$$\sum_{i=1}^{N} (Zx_i)^t \, (Wx_i - y_i) = 0$$

rewritten as $\sum_{i=1}^{N} \langle x_i \otimes (W_* x_i - y_i) : Z \rangle = 0$. Since Z is arbitrary one gets $\sum_{i=1}^{N} x_i \otimes (W_* x_i - y_i) = 0$. More transformations yield

$$\sum_{i=1}^{N} x_i \, (W_* x_i - y_i)^t = \sum_{i=1}^{N} x_i x_i^t W^t - x_i y_i^t = \left(\sum_{i=1}^{N} x_i \otimes x_i \right) W^t - \sum_{i=1}^{N} x_i \otimes y_i = 0.$$

Assume the linear independence of the vectors x_i. It yields that the matrix $\left(\sum_{i=1}^N x_i \otimes x_i\right)$ is non singular that is it is invertible. One gets $W^t = \left(\sum_{i=1}^N x_i \otimes x_i\right)^{-1} \sum_{i=1}^N x_i \otimes y_i$ and

$$W = \left(\sum_{i=1}^N y_i \otimes x_i\right) \left(\sum_{i=1}^N x_i \otimes x_i\right)^{-1} \in \mathcal{M}_{nm}(\mathbb{R}).$$

The function $h(\varepsilon) = J(W+\varepsilon Z)$ is a polynomial with respect to ε, with degree $\deg(h) \leqslant 2$. Therefore

$$h(\varepsilon) = h(0) + h'(0)\varepsilon + \frac{1}{2}h''(0)\varepsilon^2 = h(0) + \frac{1}{2}\left(\sum_{i=1}^N |Zx_i|^2\right)\varepsilon^2 \geqslant h(0)$$

because $h'(0) = 0$ by definition of the matrix W. That is

$$J(W + \varepsilon Z) \geqslant J(W)$$

for all possible ε and all Z. So W is indeed a minimum of the function J.

Second step. Now b can be non zero. Let us make the correspondence for matrices

$$\widehat{W} = (W \mid b) \in \mathcal{M}_{n,m+1}(\mathbb{R}) \tag{7}$$

and the correspondance for vectors

$$\widehat{x}_i = \begin{pmatrix} x_i \\ 1 \end{pmatrix}. \tag{8}$$

By construction $\widehat{W}\widehat{x}_i = Wx_i + b$. Therefore one can set a new function

$$\widehat{J}(\widehat{W}) = \frac{1}{2}\sum_{i=1}^N \left|\widehat{W}\widehat{x}_i - y_i\right|^2 = \frac{1}{2}\sum_{i=1}^N |Wx_i + b - y_i|^2. \tag{9}$$

The first step yields the formula for the optimal solution

$$(W, b) = \left(\sum_{i=1}^N y_i \otimes \widehat{x}_i\right) \left(\sum_{i=1}^N \widehat{x}_i \otimes \widehat{x}_i\right)^{-1} \in \mathcal{M}_{n,m+1}(\mathbb{R}).$$

Final comment. So far, the proof assumed the linear independence of the x_i, which is useful to construct W solution of $WM = B$ with $M = M^T = \left(\sum_{i=1}^N x_i \otimes x_i\right)$ and $B = \sum_{i=1}^N y_i \otimes x_i$. It is a classical exercise in linear algebra to show that, even if the x_i do not span the entire space, there still exists a solution W^t which is nevertheless non unique. To prove this standard result we use a penalization method which can be implemented. We take a small number $\varepsilon > 0$ and we consider

$$W_\varepsilon M_\varepsilon = B, \qquad M_\varepsilon = M_\varepsilon^T = \left(\sum_{i=1}^N x_i \otimes x_i\right) + \varepsilon I_m.$$

Then $\|W_\varepsilon\| = \sup_{\|y\|=1, \|x\|=1} \langle y, W_\varepsilon x\rangle$ with $W_\varepsilon x = Bz_\varepsilon$ and $\left(\left(\sum_{i=1}^N x_i \otimes x_i\right) + \varepsilon I_m\right) z_\varepsilon = x$. Consider the orthogonal decomposition $\mathbb{R}^m = V \oplus V^\perp$ where $V = \operatorname{Span}_{1 \leqslant i \leqslant N}(x_i)$.

• For $x \in V^\perp$, one checks that $z_\varepsilon = \frac{1}{\varepsilon}x$ and $Bz_\varepsilon = 0$. In this case, $\langle y, W_\varepsilon x\rangle = 0$.
• For $x \in V$, we quote the equivalence of norms in V $\alpha\|x\|_N \leqslant \|x\| \leqslant \beta\|x\|_N$ where $\|x\|_N = \sqrt{\sum_{i=1}^N \langle x, x_i\rangle^2}$ and $0 < \alpha \leqslant \beta$. Then one has a bound uniform with respect to ε since

$$\|z_\varepsilon\|_N^2 \leqslant \langle M_\varepsilon z_\varepsilon, z_\varepsilon\rangle = \langle x, z_\varepsilon\rangle \leqslant \|x\|\|z_\varepsilon\| \leqslant \beta\|x\|\|z_\varepsilon\|_N \Longrightarrow \|z_\varepsilon\|_N \leqslant \beta\|x\| = \beta.$$

Finally with an intermediate Cauchy-Schwarz inequality $|\langle y, W_\varepsilon x\rangle| \leqslant |\langle y, Bz_\varepsilon\rangle| \leqslant C\|y\|\|z_\varepsilon\| \leqslant C\frac{\beta}{\alpha}$.
• Finally $\|W_\varepsilon\| \leqslant C\frac{\beta}{\alpha}$. The matrices W_ε belong to a compact set, so one can extract a sub-sequence which admits a limit denoted as W_*. □

Exercise 2.3. *Show that W_* limit of W_ε is indeed a minimizer.*

The matrix of weights W and the offset b are important tools in ML, but in combination with at least two other ingredients. The first one is **recursivity**, the second one is **non linearity**. Note that historically, non linearity was introduced by Rosenblatt for the construction of the **perceptron** before the use of recursivity. However the mathematical structure is perhaps better highlighted with the presentation in the reverse order, that recursivity before non linearity.

2.1.1 Recursivity

Let us first describe recursivity. One takes a sequence of integers

$$(a_0, a_1, a_2, \dots, a_p, a_{p+1}) \in \mathbb{N}^{p+1}$$

such that the first one is $a_0 = m$ and the last one is $a_{p+1} = m$, and a sequence of matrices of weights and offsets

$$W_r \in \mathcal{M}_{a_{r+1},a_r}(\mathbb{R}) \quad \text{and } b_r \in \mathbb{R}^{a_{r+1}}, \qquad 0 \leqslant r \leqslant p. \tag{10}$$

A generalization of the least square example is based on a sequence of linear functions

$$\begin{array}{rccc} f_r: & \mathbb{R}^{a_r} & \longrightarrow & \mathbb{R}^{a_{r+1}}, \\ & X_r & \longmapsto & f_r(X_r) = W_r X_r + b_r. \end{array} \tag{11}$$

To introduce recursivity, a first but naive possibility is to consider the function

$$\begin{array}{rccc} f: & \mathbb{R}^m & \longrightarrow & \mathbb{R}^n, \\ & X & \longmapsto & f(X) = f_p(f_{p-1}(\dots(f_2(f_1(f_0(X)))\dots)). \end{array} \tag{12}$$

The result is also noted as a composition

$$f = f_p \circ f_{p-1} \dots f_2 \circ f_1 \circ f_0. \tag{13}$$

One could think that, just because there is more degrees of freedom to optimize, then this formula is potentially more accurate. This is just the opposite, and actually the result is equal to a linear function.

Lemma 2.4. *The function (13) is equivalent to the simple least square approximation (5) with*

$$W = W_{p+1}W_{p-1}\dots W_2W_1W_0 \in \mathcal{M}_{nm}(\mathbb{R})$$

and

$$b = b_{p+1} + \sum_{r=0}^{p} W_{p+1}\dots W_{r+1}b_r \in \mathbb{R}^n.$$

Nevertheless, with non linearity, recursivity opens wonderful new possibilities.

2.1.2 Non linearity

In order to enrich the construction and to break this linear degeneracy, one introduces a non linear ingredient which is called an **activation function**[2]. Following the standard notations, we consider either a sigmoid function σ or the rectified linear unit function R.

Definition 2.5. *An continuously differentiable function $\sigma \in C^1(\mathbb{R})$ is called a sigmoid if $\sigma' > 0$, if it has the limit value 0 at $-\infty$ and if it has the value 1 at $+\infty$. For practical reasons, the sigmoid will be more denoted with the letter S (that is $S = \sigma$).*

By definition $0 = S(-\infty) \leqslant S(x) \leqslant S(+\infty) = 1$.

Definition 2.6. *The rectified linear unit function (ReLU) $R \in C^0(\mathbb{R})$ is continuous and is defined by*

$$R(x) = \frac{1}{2}(x + |x|) = x_+ = \max(x, 0) = \begin{cases} 0 & for \quad x \leqslant 0, \\ x & for \quad 0 \leqslant x. \end{cases}$$

Definition 2.7. *The rectified linear unit function with threshold $T \in C^0(\mathbb{R})$ is continuous and is defined*

$$T(x) = \min(R(x), 1) \in [0, 1].$$

An useful abuse of notation is to apply the sigmoid or the ReLU component wise to any entries of a vector (or a matrix). That is, for $X = (x_1, \ldots, x_m)^t \in \mathbb{R}^m$,

$$h(X) = (h(x_1), \ldots, h(x_m))^t \in \mathbb{R}^m \quad \text{where } h = R, S \text{ or } T.$$

Then, a standard way to introduce our last ingredient, **non linearity**, in the design of function is to consider instead of (13)

$$f = f_p \circ g_{p-1} \cdots \circ g_2 \circ g_1 \circ g_0, \qquad \text{with } g_r = h_r \circ f_r \text{ where } h_r = R, S \text{ or } T. \tag{14}$$

That is one intertwins activation functions with linear functions.

Definition 2.8. *Let $f_0 : \mathbb{R}^n \longrightarrow \mathbb{R}$ be affine. Then $g = S \circ f_0$ is called a Rosenblatt perceptron (1958) with n inputs.*

It is valuable to present the notation with graphs, as in `http://alexlenail.me/NN-SVG/index.html`. The input layer is $\mathbb{R}^m$. The output layer is $\mathbb{R}^n$. All other layers of size $a_1, \ldots, a_p$ are called hidden layers, that is with our notations, p is the number of hidden layers. Moreover one calls them dense layers because all entries of the matrices of weights are degrees of freedom. The graphs must be read from the left to the right and they explain the flowchart of the data.

From now one all matrices of weights and vectors of offsets are denoted with one letter W. It is a super-matrix of weights

$$W = (W_{p+1}, b_{p+1}, W_{p-1}, b_{p-1}, \ldots, W_0, b_0) \in \mathcal{M} \tag{15}$$

where the dimension of $\mathcal{M}$ is grows as

$$\dim(\mathcal{M}) = \sum_{r=0}^{p+1} (a_r + 1)a_{r+1}. \tag{16}$$

[2]There is whole zoo of different activation functions to play with which already implemented in softwares.

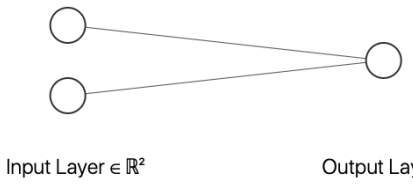

Figure 1: Graph structure of the perceptron. The input layer has dimension $m = 2$. The output layer has dimension $n = 1$. There is no hidden layer.

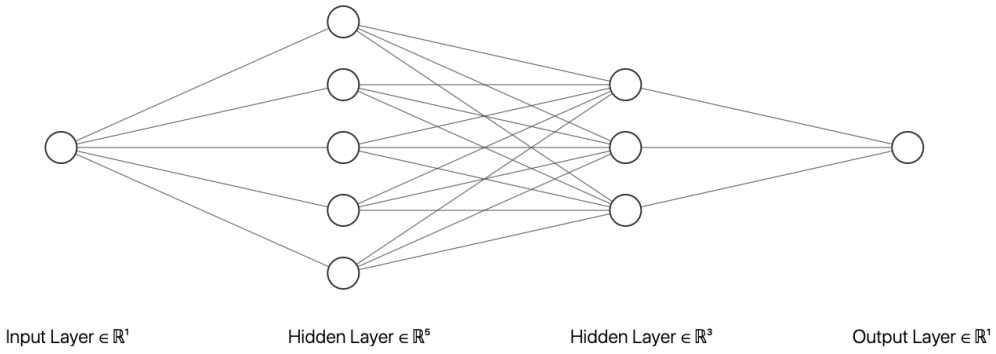

Figure 2: Graph structure of a neural network with $p = 2$ dense hidden layers, one input layer and one output layer which have the same dimension $m = 1$.

One obtains a function which can be denoted as

$$f(x, W) : \quad \mathbb{R}^m \quad \longrightarrow \mathbb{R}^n. \tag{17}$$

We immediately quote the fact that the weights and offsets can be changed without changing the function.

Lemma 2.9. *Permutations of the neurons of the hidden dense layers leave f unchanged. the number of such permutations is*

$$\#(\textit{permutations}) = a_1! \times \cdots \times a_p! \quad \textit{where } p = \#(\textit{hidden layers}).$$

Proof. Permutations of neurons keeping the connections the same (that is permuting the order with which one stores the coefficients in W_r) and keeping the offsets the same (that is permuting as well the order with which one stores the coefficients in b_r) do not change the result. □

It remains fit the parameters, that is to construct the best matrix $W \in \mathcal{M}$. A generalization of the least squares procedure considers the minimization problem

$$W_* = \underset{W}{\operatorname{argmin}} \sum_{(x,y)\in\mathcal{D}} |f(x,W) - y|^2$$

where the dataset $\mathcal{D}$ must be specified. All questions considered in this text are variations around this theme.

Does the optimal weight W_* exists? Can we calculate W_* on a computer in an accurate way? Are they stable procedures to calculate W_* ? These problems lay the foundations of modern Machine Learning and Neural Network algorithms.

2.1.3 Convolutive neural networks (CNN)

It is possible to introduce and understand the possible interest of convolutive neural network just by comparison with the structure of dense layers illustrated in Figure 2 and with the complexity formula (16) for the dimension of $\mathcal{M}$. If the number of layers is important and the width of the layers is also important, then $\dim(\mathcal{M})$ can be huge. With this regard, it is natural to try to control the dimension. For convolutive neural networks, there is a possibility which comes from symmetry principles (such as rotation of translation principles) which are satisfied for example for image classification.

Independently of the motivations, theis procedure can be describe in terms of good sparse structure of the matrices, such as Toeplitz matrices, band matrices or convolution matrices (so the name[3]).

Definition 2.10. *A square Toeplitz matrix $W \in \mathcal{M}_a(\mathbb{R})$ is such that*

$$w_{i,j} = s\,(i+j) \qquad 1 \leqslant i,j \leqslant a.$$

Definition 2.11. *A square band matrix $W \in \mathcal{M}_a(\mathbb{R})$ of size $1 \leqslant d \leqslant a/2$ is such that*

$$w_{i,j} = 0 \text{ for } |i-j| \geqslant d.$$

For such square Toeplitz band matrices, the number of degrees of freedom (of free parameters) is equal to $2d-1$ which can be much less than a^2. So if the final accuracy of the function f encoded in (14) is enough, it is a good option to dramatically reduce the number of parameters to adapt in W.

A natural extension in higher dimension needed for classification of images (LeCun) can be written as follows. For that we consider that the entries (the components) of a vector $x \in \mathbb{R}^a$ correspond to the storage of the pixels of an image. Sweeping along the $q \geqslant 1$ dimensions of the image yields the tensor (in this text, it is just a matrix with multi-indices)

$$y \in \mathcal{M}_{\alpha_1,\dots,\alpha_q}(\mathbb{R}), \qquad b = \alpha_1 \dots \alpha_q, \tag{18}$$

[3]Toeplitz band matrices are discrete version of convolutions. Here convolution is understood as follows. One starts with two functions f and g define on the real line $\mathbb{R}$. The function $h = w * g$ is the convolution of w and g

$$h(x) = \int_{\mathbb{R}} w(x-y)g(y)dy = \int_{\mathbb{R}} w(y)g(x-y)dy.$$

If the kernel f has a compact support, it ressembles a Toeplitz matrix with a band structure. Since convolution kernels are quite common in nature and science, it explains the importance of this topic.

with the correspondance $x_i = y_{\beta_1,\dots,\beta_q}$ where the indices are related[4] by

$$j = \beta_1 + \alpha_1(\beta_2 - 1) + \alpha_1\alpha_2(\beta_3 - 1) + \cdots + \alpha_1 \times \cdots \times \alpha_{q-1}(\beta_q - 1) \tag{19}$$

and the bounds $1 \leqslant \beta_r \leqslant \alpha_r$ for $1 \leqslant r \leqslant q$ and $1 \leqslant j \leqslant b$. One checks that (19) is a bijection between the two different ways of indexing, in notation

$$j = (\beta_1, \dots, \beta_q). \tag{20}$$

Definition 2.12. *A square convolutive*[5] *matrix $W \in \mathcal{M}_b(\mathbb{R})$ associated to (18-19) is defined by*

$$w_{i,j} = s\left(\beta_1 - \gamma_1, \dots, \beta_q - \gamma_q\right) \qquad 1 \leqslant i, j \leqslant b \tag{21}$$

where the indices are defined by (19) and similarly by

$$i = \gamma_1 + \alpha_1(\gamma_2 - 1) + \alpha_1\alpha_2(\gamma_3 - 1) + \cdots + \alpha_1 \times \cdots \times \alpha_{q-1}(\gamma_q - 1)$$

and the bounds $1 \leqslant \gamma_s \leqslant \alpha_s$ for $1 \leqslant s \leqslant q$ and $1 \leqslant j \leqslant b$.

Even if the structure is now adapted to the natural storage of pixels in images, more savings of computational effort is still possible. The basic idea is that it is probably not necessary to use the full matrix (21) for all i and j. Typically, in a real image, the result $y = Wx$ can be very close for neighboring indices of the vector x (neighboring using the storage provided by formulae (19).

To implement the idea, a possibility is to consider an additional multiplicative decomposition of α_r for $1 \leqslant r \leqslant q$ as follows

$$\alpha_r = \overline{\alpha}_r\widehat{\alpha}_r, \quad \overline{\alpha}_r, \widehat{\alpha}_r \in \mathbb{N}^*.$$

Now (19) is complemented by

$$\beta_r = \widehat{\beta}_r + \widehat{\alpha}_r(\overline{\beta}_r - 1) \tag{22}$$

with the bounds $1 \leqslant \widehat{\beta}_r \leqslant \widehat{\alpha}_r$ and $1 \leqslant \overline{\beta}_r \leqslant \overline{\alpha}_r$ or $1 \leqslant r \leqslant q$.

We note $b = \overline{\alpha}_1 \dots \overline{\alpha}_q$ which is strictly smaller than a if just one $\overline{\alpha}_r$ is greater or equal to 2. Let us define

$$i = \overline{\beta}_1 + \overline{\alpha}_1(\overline{\beta}_2 - 1) + \overline{\alpha}_1\overline{\alpha}_2(\overline{\beta}_3 - 1) + \cdots + \overline{\alpha}_1 \times \cdots \times \overline{\alpha}_{q-1}(\overline{\beta}_q - 1), \qquad 1 \leqslant i \leqslant a = \overline{\alpha}_1 \times \cdots \times \overline{\alpha}_q$$

with bounds $1 \leqslant \overline{\beta}_s \leqslant \overline{\alpha}_s$ for $1 \leqslant s \leqslant q$. One can perform the same identification as in (20)

$$i = (\overline{\beta}_1, \dots, \overline{\beta}_q).$$

[4]This formula is commonly used in Finite Difference methods for reindexing the indices, from a multi-entry index to a single index.

[5]Same remark as before. Start from

$$h(\mathbf{x}) = \int_{\mathbb{R}} w(\mathbf{x} - \mathbf{y})g(\mathbf{y})dy = \int_{\mathbb{R}} w(\mathbf{y})g(\mathbf{x} - \mathbf{y})dy,$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ with $d \geqslant 1$. Assume w has a compact support $|\mathbf{x}| \geqslant R \Longrightarrow w(\mathbf{x}) = 0$. So one can write

$$h(x_1, \dots, x_d) = \int_{(-R,R)^d} w(x_1 - y_1, \dots, x_d - y_d)g(y_1, \dots, y_d)dy_1 \dots dy_d.$$

After sampling along the coordinates, one can discretize this operation with a convolutive matrix.

Definition 2.13. *A rectangular convolutive matrix $W \in \mathcal{M}_{ab}(\mathbb{R})$ with stride is defined by*

$$w_{i,j} = s\left(\overline{\beta}_1, \ldots, \overline{\beta}_q\right) \qquad 1 \leqslant i \leqslant a,\ 1 \leqslant j \leqslant b. \tag{23}$$

Now the size of the vector $y = Wx$ is a which is less than b the size of x. With these respect it is possible to consider

Definition 2.14. *A convolutive tensor $W \in \mathcal{M}_{a,c,b}(\mathbb{R})$ with stride is defined by*

$$w^k_{i,j} = s^k\left(\overline{\beta}_1, \ldots, \overline{\beta}_q\right) \qquad 1 \leqslant i \leqslant a,\ 1 \leqslant k \leqslant c,\ 1 \leqslant j \leqslant b = ac. \tag{24}$$

Further compression of a convolutive tensor $W \in \mathcal{M}_{a,c,b}(\mathbb{R})$ to obtain for example a convolutive tensor $W \in \mathcal{M}_{a,d,b}(\mathbb{R})$ with $d < c$ is usually performed with the maxpool function [4]. These technical details are important for image classification.

2.2 Logistic function, cross-entropy, classification

The great success of neural networks is for classification, for which least squares inspired methods can perform bad. In classification, the output y is a discrete probability, that is

$$y \in \mathcal{P}_n = \{y \in [0,1]^n, y_1 + \ldots y_n = 1\} \underset{\neq}{\subset} [0,1]^n \subset \mathbb{R}^n.$$

This is a discrete probability space, with n classes. Think of image classification (an image, that is a collection of pixels, corresponds to a cat or a dog with probability 1 ou 0). Of course it is possible in principle to continue the principles already exposed, for example using just a least approximation. Not only it might be quite naive (why not using the full information on y being a discrete probability ?), but the numerical results in the literature for classification are indubitably in favor of using the information that $y \in \mathcal{P}_n$.

2.2.1 Logistic function, cross-entropy

The idea is to replace the least squares approximation by what is called logistic regression. Consider a function $f : \mathbb{R}^m \to \mathbb{R}^n$, modeled through a representation like (14). Consider $z = f(x) \in \mathbb{R}^n$. Two definitions are enough to present the most important part of this approach.

Definition 2.15. *For any $n \geqslant 1$, the logistic function is*

$$\begin{array}{rccl} \text{Logistic}: & \mathbb{R}^n & \to & \mathcal{P}_n \\ & z & \mapsto & p = (p_1, \ldots, p_n), \quad \textit{with } p_r = \frac{e^{z_r}}{\sum_{s=1}^n e^{z_s}}. \end{array} \tag{25}$$

The logistic function transforms any vector in a probability vector.

Lemma 2.16. *One has (evident)*

$$\text{Logistic}(z + \lambda e) = \text{Logistic}(z) \textit{ for } e = (1, \ldots, 1) \in \mathbb{R}^n \textit{ and for all } \lambda \in \mathbb{R}.$$

Definition 2.17. *For any $n \geqslant 1$, the cross-entropy function is*

$$\begin{array}{rccl} \text{Cross}: & \mathbb{R}^n \times \mathcal{P}_n & \to & \mathbb{R}^+ \\ & (z, y) & \mapsto & -\sum_{s=1}^n \log(p_s) y_s \geqslant 0, \quad \textit{with } p = \text{Logistic}(z). \end{array}$$

The cross-entropy is related the Kullback-Leibler divergence (KL divergence), also denoted as the relative entropy

$$KL(y,p) = \sum_{s=1}^{n} y_s \log \frac{y_s}{p_s} = \sum_{s=1}^{n} y_s \log y_s - \mathrm{Cross}(z,y).$$

One has

$$\mathrm{Cross}(z,y) = -\sum_{s=1}^{n} z_s y_s + \log\left(\sum_{s=1}^{n} e^{z_s}\right). \tag{26}$$

Lemma 2.18. *The cross-entropy is convex with respect to its first variable $z \in \mathbb{R}^n$. For a given $y \in (0,1)^n$, there exists a minimum $z_* \in \mathbb{R}^n$*

$$\mathrm{Cross}(z_*, y) \leqslant \mathrm{Cross}(z,y) \text{ for all } z \in \mathbb{R}^n$$

which is characterized by $\mathrm{Logistic}(z_*) = y$. *The minimum is given by an explicit formula and is unique up to the addition of λe for any $\lambda \in \mathbb{R}$ (and $e = (1,\dots,1) \in \mathbb{R}^p$).*

Proof. With (26), one has for $z, y \in \mathbb{R}^n$

$$\nabla_z \mathrm{Cross}(z,y) = \mathrm{logis}(z) - y. \tag{27}$$

We will also use the notation $p = \mathrm{logis}(z) \in \mathcal{P}_n$. The rest of the proof is as follows.
• The Hessian is the symmetric matrix $\nabla_z^2 \mathrm{Cross}(z,y) = \nabla_z \mathrm{logis}(z)$, that is after calculations

$$\nabla_z^2 \mathrm{Cross}(z,y) = \mathrm{diag}(p_r) - (p_r p_s)_{1\leqslant r,s\leqslant n} \in \mathcal{M}_n(\mathbb{R}).$$

Using that $\sum_r p_r = 1$, one obtains $\nabla_z^2 \mathrm{Cross}(z,y) = \mathrm{diag}\left(p_r \sum_{s\neq r} p_s\right) - (p_r p_s)_{1\leqslant r\neq s\leqslant n}$ which is a non negative symmetric matrix. Let $d = (d_r)_{1\leqslant r\leqslant n} \in \mathbb{R}^n$ be a test direction. Then

$$\left\langle \nabla_z^2 \mathrm{Cross}(z,y)d, d\right\rangle = \sum_{1\leqslant r\neq s\leqslant n} p_r p_r (d_r - d_s)^2 \geqslant 0. \tag{28}$$

Since it holds for all $d \in \mathbb{R}^n$, the Hessian is non negativity, and so the function is convex with respect to z.
• At a minimum, the gradient vanishes. It writes $\frac{e^{z_r}}{\mu} = y_r$ with $\mu = \sum_{s=1}^n e^{z_s} > 0$. So $z_r = \log(y_r \mu)$. It is correctly defined only for $y_r > 0$: this is the reason of the restriction $y \in (0,1)^n$. One can write $z_r = \log y_r + \lambda$, with $\lambda = \log \mu$. Taking $\mu = 1$, it defines

$$z_* = (\log y_r)_{1\leqslant r\leqslant n}. \tag{29}$$

Since the function is convex and the gradient vanishes at z_*, it means that z_* is a minimum given by an explicit formula.
• Since the probabilities are positive, then

$$\ker\left\{d \mapsto \left\langle \nabla_z^2 \mathrm{Cross}(z,y)d, d\right\rangle\right\} = e\mathbb{R}, \quad e = (1,\dots,1) \in \mathbb{R}^n$$

which yields the last part of the claim. □

If one of the y_i's vanishes, then there is no more a minimum but an infimum.

2.2.2 Classification

One implements the logistic regression over the dataset $\mathcal{D}$ by summation

$$J(W) = \sum_{i=1}^{N} \mathrm{Cross}(Wx_i + b, y_i) \geqslant 0. \tag{30}$$

Still with the convention $\widehat{W}\widehat{x}_i = Wx_i + b$ already introduced in (9), it is also noted $\widehat{J}(\widehat{W}) = \sum_{i=1}^{N} \mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i)$. An expansion yields

$$\widehat{J}(\widehat{W}) = -\sum_{i=1}^{N}\sum_{r=1}^{m} \log\left(\frac{e^{\left\langle \widehat{W}\widehat{x}_i, e_r\right\rangle}}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}\widehat{x}_i, e_s\right\rangle}}\right) y_i^r, \qquad y_i^r = \langle y_i, e_r\rangle\,.$$

Lemma 2.19. *The first derivative of $\widehat{J}$ with respect to $\widehat{W}$ is*

$$\nabla_{\widehat{W}}\widehat{J}(\widehat{W}) = \sum_{i=1}^{N} \left(\mathrm{Logistic}\left(\widehat{W}\widehat{x}_i\right) - y_i\right) \otimes \widehat{x}_i. \tag{31}$$

The function $\widehat{J}$ is convex since

$$\left\langle \nabla^2_{\widehat{W}} J(W)\overline{W}, \overline{W}\right\rangle = \sum_{i=1}^{N} \left(\nabla_z^2 \mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i)\overline{W}\widehat{x}_i, \overline{W}\widehat{x}_i\right) \geqslant 0, \quad \forall \overline{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}).$$

The function $\widehat{J}$ is not strictly convex since this quantity vanishes for $\overline{W} = \lambda(0_{n,m}, e)$.

Proof. The first derivative comes from the summation of

$$\nabla_{\widehat{W}}\mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i) = \nabla z \mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i) \otimes \widehat{x}_i = \left(\mathrm{Logistic}\left(\widehat{W}\widehat{x}_i\right) - y_i\right) \otimes \widehat{x}_i.$$

The Hessian matrix comes from the summation of

$$\left\langle \nabla^2_{\widehat{W}} \mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i)\overline{W} : \overline{W}\right\rangle = \left\langle \nabla_z^2 \mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i)\overline{W}\widehat{x}_i, \overline{W}\widehat{x}_i\right\rangle$$

for all test matrices $\overline{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$. The non negativity is a consequence of (28). □

Our objective is to find a minimizer $\widehat{W}_* \in \mathcal{M}_{n,m+1}(\mathbb{R})$ such that $0 \leqslant \widehat{J}(\widehat{W}_*) \leqslant \widehat{J}(\widehat{W})$ for all $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$. However no explicit solution of the Euler-Lagrange equation $\nabla\widehat{J}(\widehat{W}_*) = 0$ is known because the gradient (31) has a complicated structure. However it is possible to prove that a (non unique) minimum exists, using standard methods in convex analysis.

A preliminary remark is that the infimum of $\widehat{J}$ is necessarily lower than $\widehat{J}(\widehat{0}) = N\log n > 0$. So it is sufficient to consider a smaller set

$$\mathcal{K}_0 = \left\{\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}), \widehat{J}(\widehat{W}) \leqslant N\log n\right\}$$

and to look for a minimum $\widehat{W}_* \in \mathcal{K}_0$ such that

$$0 \leqslant \widehat{J}(\widehat{W}_*) \leqslant \widehat{J}(\widehat{W}) \text{ for all } \widehat{W} \in \mathcal{K}_0. \tag{32}$$

We consider the scalar product between matrices

$$\left\langle \widehat{W}, \widehat{Z} \right\rangle = \sum_{\alpha=1}^{n} \sum_{\beta=1}^{m+1} \widehat{w}_{\alpha\beta} \widehat{z}_{\alpha\beta} \in \mathbb{R}, \qquad \widehat{W}, \widehat{Z} \in \mathcal{M}_{n,m+1}(\mathbb{R}).$$

Definition 2.20. *Let us define the vectorial subspace*

$$\mathcal{M} = \underset{1 \leqslant i \leqslant N,\ 2 \leqslant r \leqslant m}{Span} \{(e_r - e_1) \otimes \widehat{x}_i\}$$

and the vectorial subspace

$$\mathcal{L} = \left\{ \widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}), \left(\widehat{W} \widehat{x}_i, e_r - e_1 \right) = 0 \text{ for all } 1 \leqslant i \leqslant N \text{ and } 2 \leqslant r \leqslant m \right\} \subset \mathcal{M}_{n,m+1}(\mathbb{R}).$$

These spaces are orthogonal, that is $\mathcal{L} = \mathcal{M}^{\perp}$ *and* $\mathcal{M} = \mathcal{L}^{\perp}$.

Lemma 2.21. *One has* $\widehat{J}(\widehat{W}) = J(\widehat{W}_2)$ *with the orthogonal decomposition of* $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$

$$\widehat{W} = \widehat{W}_1 + \widehat{W}_2, \quad \widehat{W}_1 \in \mathcal{M},\ \widehat{W}_2 \in \mathcal{L}.$$

Proof. By definition, one has

$$\widehat{J}(\widehat{W}_1 + \widehat{W}_2) = -\sum_{i=1}^{N} \sum_{r=1}^{m} \log \left(\frac{e^{\left\langle \widehat{W}_1 \widehat{x}_i, e_r \right\rangle} e^{\left\langle \widehat{W}_2 \widehat{x}_i, e_r \right\rangle}}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}_1 \widehat{x}_i, e_s \right\rangle} e^{\left\langle \widehat{W}_2 \widehat{x}_i, e_s \right\rangle}} \right) y_i^r.$$

The condition $\widehat{W}_2 \in \mathcal{L}$ yields the simplification

$$\widehat{J}(\widehat{W}_1 + \widehat{W}_2) = -\sum_{i=1}^{N} \sum_{r=1}^{m} \log \left(\frac{e^{\left\langle \widehat{W}_1 \widehat{x}_i, e_r \right\rangle}}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}_1 \widehat{x}_i, e_s \right\rangle}} \right) y_i^r = J(\widehat{W}_1).$$

□

Theorem 2.22. *Assume* $y_i \in (0,1)^n$ *for all* $1 \leqslant i \leqslant N$. *Then there exists a (non unique) matrix* $\widehat{W}_* \in \mathcal{M}_{n,m+1}(\mathbb{R})$ *such that of* $\widehat{J}(\widehat{W}_*) \leqslant \widehat{J}(\widehat{W})$ *for all* $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$.

Proof. The function

$$\begin{array}{rccc} \widehat{J}^{\perp} : & \mathcal{M} & \to & \mathbb{R}, \\ & \widehat{W} & \to & \widehat{J}(\widehat{W}) \end{array}$$

is continuous. Let us take a sufficiently large $K > 0$ and consider the set

$$S(K) = \left\{ \sup_{i=1}^{N} \sup_{r=2}^{m} \left| \left(\widehat{W} \widehat{x}_i, e_1 - e_r \right) \right| \leqslant K \right\}.$$

Using Lemma 2.24, the set $S(K) \cap \mathcal{M}$ is compact. So the continuous function $\widehat{J}^{\perp}$ has a minimum over $S(K) \cap \mathcal{M}$. We denote this minimum as $\widehat{W}_* \in S(K) \cap \mathcal{M}$.

Using Lemma 2.21, $\widehat{W}_*$ is also a minimizer of $\widehat{J}$ over $S(K)$

$$\widehat{J}(\widehat{W}_*) \leqslant \widehat{J}(\widehat{W}), \quad \forall \widehat{W} \in S(K).$$

Using Lemma 2.23, it is a minimizer over the whole space $\mathcal{M}_{n,m+1}(\mathbb{R})$. □

Lemma 2.23. *Assume $y_i \in (0,1)^n$ for all $1 \leqslant i \leqslant N$. Then there exists a constant $K > 0$ such that $\mathcal{K}_0 \subset S(K)$.*

Proof. Let $\widehat{W} \in \mathcal{K}_0$ and take an arbitrary $1 \leqslant l \leqslant N$. One has

$$0 \leqslant -\sum_{r=1}^{m} \left\langle \text{Logistic}(\widehat{W}\widehat{x}_l), e_r \right\rangle y_l^r = \text{Cross}(\widehat{W}\widehat{x}_l, y_l) \leqslant \sum_{i=1}^{N} \text{Cross}(\widehat{W}\widehat{x}_i, y_i) = \widehat{J}(\widehat{W}) \leqslant \widehat{J}(0).$$

Take an arbitrary component $1 \leqslant r \leqslant m$. One deduces the upper bound

$$-\left\langle \text{Logistic} < \widehat{W}\widehat{x}_l), e_r \right\rangle y_l^r \leqslant \widehat{J}(0).$$

By assumption $y_l^r = (y_l, e_r) > 0$ for all l and r. Therefore there exists a small constant $\varepsilon > 0$ (independent of l and r) such that

$$\left\langle \text{Logistic}(\widehat{W}\widehat{x}_l), e_r \right\rangle \geqslant \varepsilon > 0, \quad \forall l, r.$$

It can be rewritten as

$$\frac{e^{\left\langle \widehat{W}\widehat{x}_l, e_r \right\rangle}}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}\widehat{x}_l, e_s \right\rangle}} = \frac{1}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}\widehat{x}_l, e_s - e_1 \right\rangle}} \geqslant \varepsilon \qquad \text{(note that } \varepsilon < 1\text{)}.$$

So one has

$$e^{\left\langle \widehat{W}\widehat{x}_l, e_s - e_1 \right\rangle} \leqslant \frac{1}{\varepsilon}, \qquad \forall l, s.$$

Without restriction, one can assume the ordering

$$\left\langle \widehat{W}\widehat{x}_l, e_1 \right\rangle \leqslant \left\langle \widehat{W}\widehat{x}_l, e_2 \right\rangle \leqslant \cdots \leqslant \left\langle \widehat{W}\widehat{x}_l, e_m \right\rangle.$$

Therefore one gets the bound

$$0 \leqslant \left\langle \widehat{W}\widehat{x}_l, e_s - e_1 \right\rangle = \left| \left(\widehat{W}\widehat{x}_l, e_s - e_1 \right) \right| \leqslant K$$

where the constant can be taken as $K = -\log \varepsilon$. Therefore $\widehat{W} \in S(K)$. □

Lemma 2.24. *The form*

$$|||\widehat{W}||| = \sup_{i=1}^{N} \sup_{r=2}^{m} \left| \left\langle \widehat{W}\widehat{x}_i, e_1 - e_r \right\rangle \right|, \quad \widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$$

is a norm on $\mathcal{M}$.

Proof. It is sufficient to show that $\widehat{W} \in \mathcal{M}$ and $|||\widehat{W}||| = 0$ implies that $\widehat{W} = 0$.

Assume $|||\widehat{W}||| = 0$, so $\left\langle \widehat{W}\widehat{x}_i, e_1 - e_r \right\rangle = 0$ for all possible i ad r. It means that $\widehat{W} \in \mathcal{L}$. Therefore $\widehat{W} \in \mathcal{M} \cap \mathcal{L} = \{0\}$, and the proof is ended. □

In the general case, one considers

$$J(W) = \sum_{i=1}^{N} \text{Cross}(f(x_i, W), y_i) \tag{33}$$

where the parameters W and the function f are defined in (15-17).

Remark 2.25. *The restriction $y_i \in (0,1)^m$ is not a real one for classification.*

3 Approximation theory, interpretation in terms of the Finite Element Method (FEM)

In this section we present some reasons why structures like (14) can approximate a f^{obj}. This is based on approximation theory and connection with FEM, and the justification follows from accuracy estimates.

3.1 Cybenko theorem and related results

The Cybenko Theorem has an historial importance in the field. All the following arguments come for the theory of approximation. It describes neural networks with one hidden layer. Even if the activation function is the sigmoid S in the original Cybenko Theorem, we will see that simple proofs of it can be made also with R as an activation function. Our goal here is to be as elementary as possible. We note $I_d = [0,1]^d$.

Theorem 3.1 (Cybenko [5])**.** *Let S be any continuous sigmoidal function. Then finite sums of the form*

$$G(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j S\left(\mathbf{y}_j^t \mathbf{x} + \theta_j\right) \tag{34}$$

are dense in $C^0(I_d)$ equipped with the norm of the maximum.

We start by constructive proofs of elementary variants of the Cybenko Theorem[6], all based on special integral representations. One more interested by much more powerful results in relation with advanced analysis and the representation Theorem of Kolmogorov can consult the works by [9, 14, 11] and many more. A recent work [2] mangaed to propose a fully constructive proof of the Kolmogorov representation Theorem.

3.1.1 Approximation in dimension one

Take a smooth real function $f^{\text{obj}} \in C^1(\mathbb{R})$ with compact support in $I = [0,1]$. Note the Heaviside function $H(x) = 1$ for $x > 0$ and $H(x) = 0$ otherwise. One has the formula

$$f^{\text{obj}}(x) = \int_{-\infty}^{x} (f^{\text{obj}})'(y)dy = \int_{-\infty}^{+\infty} (f^{\text{obj}})'(y)H(x-y)dy. \tag{35}$$

We want to consider approximations with sigmoid functions. Regularize the Heaside function with a mollifier $\rho \in C^0(\mathbb{R}) \cap L^1(\mathbb{R})$, with $0 \leqslant \rho(x) \leqslant \rho(0) = 1$ and $\int_{\mathbb{R}} \rho(z)dz = 1$. The regularized Heaside function is $H_\varepsilon = \rho_\varepsilon * H$

$$H_\varepsilon(x) = \frac{1}{\varepsilon}\int_{\mathbb{R}} \rho\left(\frac{x-y}{\varepsilon}\right) H(y)dy = \frac{1}{\varepsilon}\int_{\mathbb{R}} \rho\left(\frac{y}{\varepsilon}\right) H(x-y)dy.$$

It yields a family of sigmoids

$$S_\varepsilon(x) = H_\varepsilon(x) = \frac{1}{\varepsilon}\int_{-\infty}^{x} \rho\left(\frac{y}{\varepsilon}\right) dy, \qquad \varepsilon > 0. \tag{36}$$

[6]The proof in [5] uses abstract functional analysis, however the key Lemma 1 page 307 is based on the fact that rescaled sigmoids approximates the Heaviside function. It is the same fundamental properties which is used below.

We note $S = \sigma_1$ which verifies

$$0 \leqslant S'(x) \leqslant S'(0) = 1 \tag{37}$$

and

$$S_\varepsilon(x) = S\left(\frac{x}{\varepsilon}\right).$$

Lemma 3.2. *The function (36) satisfies the assumptions of Definition 2.5.*

By convolution

$$f_\varepsilon^{\mathrm{obj}}(x) = (\rho_\varepsilon * f^{\mathrm{obj}})(x) = \int_{\mathbb{R}} (f^{\mathrm{obj}})'(y) S\left(\frac{x-y}{\varepsilon}\right) dy = \int_{|x|\leqslant 1} (f^{\mathrm{obj}})'(y) S\left(\frac{x-y}{\varepsilon}\right) dy, \quad x \in \mathbb{R}. \tag{38}$$

A Riemann integration procedure with step size $h = \frac{1}{N} \in (0,1]$ yields the function f_N

$$f_{\varepsilon,N}(x) = \sum_{|ih|\leqslant 1} (f^{\mathrm{obj}})'(ih) S\left(\frac{x-ih}{\varepsilon}\right) h. \tag{39}$$

Lemma 3.3. . *The function f_N can be realized with a neural network with $m = n = 1$ and with one hidden layer with $2N-1$ neurons.*

Proof. For the hidden layer, take $W_0 = \frac{1}{\varepsilon}(1,\dots,1)^t \in \mathbb{M}_{N,1}(\mathbb{R})$ proportional to the identity matrix and $b_0 = -W_0$. For the output layer, take $W_1 = \left((f^{\mathrm{obj}})'(ih)\right)_{|ih|\leqslant 1} \in \mathbb{M}_{1,N}(\mathbb{R})$ and $b_1 = 0$ which is a real number. The activation function is S for the hidden layer. The activation function of output layer is linear (it means no activation function). □

Lemma 3.4. *Assume $f^{\mathrm{obj}} \in C_0^0(I)$ and the sigmoid activation function is twice differentiable $S \in W^{2,\infty}(\mathbb{R})$. Then*

$$\lim \left\| f^{\mathrm{obj}} - f_N \right\|_{L^\infty(I)} = 0 \text{ as } N \to \infty \text{ and } \varepsilon \to 0.$$

Proof. The triangular inequality is $\|f^{\mathrm{obj}} - f_N\|_{L^\infty(I)} \leqslant \left\|f^{\mathrm{obj}} - f_\varepsilon^{\mathrm{obj}}\right\|_{L^\infty(I)} + \left\|f_\varepsilon^{\mathrm{obj}} - f_N\right\|_{L^\infty(I)}$. First

$$\left\|f^{\mathrm{obj}} - f_\varepsilon^{\mathrm{obj}}\right\|_{L^\infty(I)} \to 0 \text{ as } \varepsilon \to 0$$

by standard uniform continuity estimates on closed on a close interval. Second by standard quadrature estimates, $\left\|f_\varepsilon^{\mathrm{obj}} - f_N\right\|_{L^\infty(I)} \leqslant \|(f_\varepsilon^{\mathrm{obj}})'\|_{L^\infty(I)} N^{-1}$. But

$$f_\varepsilon^{\mathrm{obj}}(x) = \int_{\mathbb{R}} f^{\mathrm{obj}}(y) \frac{d}{dy} S\left(\frac{x-y}{\varepsilon}\right) dy \text{ so } (f_\varepsilon^{\mathrm{obj}})'(x) = \int_{\mathbb{R}} f^{\mathrm{obj}}(y) \frac{d^2}{dy^2} S\left(\frac{x-y}{\varepsilon}\right) dy$$

hence $\|(f_\varepsilon^{\mathrm{obj}})'\|_{L^\infty(I)} \leqslant \frac{C}{\varepsilon^2}$ and

$$\left\|f_\varepsilon^{\mathrm{obj}} - f_N\right\|_{L^\infty(I)} \leqslant \frac{C}{\varepsilon^2 N}.$$

Take ε small enough, then N large enough. The structure of this network is illustrated in Figure 4 □

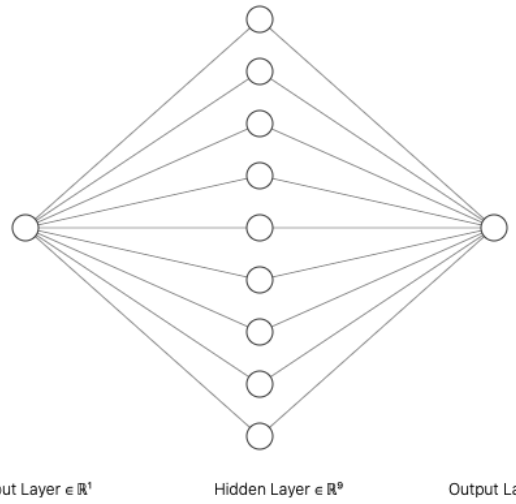

Figure 3: Graph structure corresponding to Lemma 3.3.

Remark 3.5. *The real weakness of the result is that the weights which go to ∞. If $f^{\mathrm{obj}} \in W^{1,\infty}(I)$ then W_1 is bounded. However W_0 blows up like ε^{-1}. Most of the methods which are really implemented tend on the contrary to control the magnitude of the weights (in some norm). So the theoretical result is absolutely not representative of the practice.*

Now we perform another manipulation. Since $R' = H$ (almost everywhere), an integration by part in (35) yields

$$f^{\mathrm{obj}}(x) = \int_{-1}^{1} (f^{\mathrm{obj}})''(y) R(x-y) dy.$$

A standard discretization procedure (Riemann formula with mesh size $h > 0$) yields the approximation

$$\overline{f_N}(x) = \sum_{|ih| \leqslant 1} (f^{\mathrm{obj}})''(ih) R(x - ih) h. \tag{40}$$

Lemma 3.6. *The function $\overline{f_N}$ can be realized with a neural network with $m = n = 1$, with one hidden layer with $2N - 1$ neurons and with the ReLU function as activation function.*

Proof. Similar as before but now $W_0 = (1, \dots, 1)^t \in \mathbb{M}_{N,1}(\mathbb{R})$. □

Lemma 3.7. *Assume $f^{\mathrm{obj}} \in W_0^{\infty,3}(I)$. One has $\left\| f^{\mathrm{obj}} - \overline{f_N} \right\|_{L^\infty(I)} \leqslant C \left\| f^{\mathrm{obj}} \right\|_{W^{\infty,3}(I)} N^{-1}$.*

Lemma 3.8. *Assume $f^{\mathrm{obj}} \in C_0^0(I)$. Then*

$$\liminf_{\overline{g_N}} \left\| f^{\mathrm{obj}} - \overline{g_N} \right\|_{L^\infty(I)} = 0 \text{ as } N \to \infty$$

where the $\overline{g_N}$ are of the form (40).

Proof. Consider $\overline{g_N}(x) = \sum_{|ih|\leqslant 1}(f_\varepsilon)''(ih)R(x-ih)h$ where g_ε is regularized by convolution

$$g_\varepsilon(x) = \rho_\varepsilon * f^{\text{obj}}(x) = \int_{\mathbb{R}} f^{\text{obj}}(y)\rho\left(\frac{x-y}{\varepsilon}\right)dy.$$

Here $\rho \in C_0^3(\mathbb{R})$ is a smooth convolution kernel, not necessarily related to a sigmoid. Then $\lim_{\varepsilon=0^+}\|g_\varepsilon - f^{\text{obj}}\|_{L^\infty(I)} = 0$.

Moreover $\|g_\varepsilon\|_{W^{3,\infty}(I)} \leqslant C\varepsilon^{-3}$. By Lemma 3.7, $\|g_\varepsilon - \overline{g_N}\|_{L^\infty(I)} \leqslant C\varepsilon^{-3}N^{-1}$.

To end the proof, take ε small enough, then N large enough. □

Comparison with the approximation with sigmoids show that the internal weight matrix W_0 is bounded. It can be seen as an advantage in the context of real calculations. Is it possible the change a little the ReLU function so that it ressembles more a sigmoid? The answer is positive as shown by the next Lemma.

Lemma 3.9. *One has* $f^{\text{obj}}(x) = \int_{\mathbb{R}} \alpha(y)T(x-y)dy$ *with* $\alpha(x) = \sum_{n\in\mathbb{N}}(f^{\text{obj}})''(x-n)$.

Proof. It is possible to restrict the sum to n such that $x - n \in I$. Set $\alpha = \beta'$. So

$$f^{\text{obj}}(x) = \int_{\mathbb{R}} \beta'(y)T(x-y)dy = \int_{\mathbb{R}} \beta(y)T'(x-y)dy = \int_{x-1}^{x} \beta(y)dy.$$

It yields $(f^{\text{obj}})'(x) = \beta(x) - \beta(x-1)$, so $\beta(x) = \sum_{n\in\mathbb{N}}(f^{\text{obj}})'(x-n)$. The result holds after differentiation. □

3.1.2 Approximation in higher dimensions

Similar identities hold in any dimension. Let $f^{\text{obj}} : \mathbb{R}^d \to \mathbb{R}$ be a smooth function with enough derivatives in $L^2(\mathbb{R}^d)$, that is $f^{\text{obj}} \in H_0^m(\mathbb{R}^d)$. If it is not the case, it is always possible to regularize by convolution.

Lemma 3.10. *One has the formula*

$$f^{\text{obj}}(\mathbf{x}) = \int_{\Omega\in S^d}\int_{b\in\mathbb{R}} w(\Omega, b)H(\Omega\cdot\mathbf{x} - b)d\Omega db \tag{41}$$

where $S^d = \{|\Omega| = 1\} \subset \mathbb{R}^d$ *and* $w : S^d \times \mathbb{R} \to \mathbb{R}$ *is a function given below.*

Proof. To construct w, differentiate

$$f^{\text{obj}}(\mathbf{x}) = \int_{\Omega\in S^d}\left(\int_{b<\Omega\cdot\mathbf{x}} w(\Omega, b)db\right) = \int_{\Omega\in S^d}\left(\int_{-\infty}^{\Omega\cdot\mathbf{x}} w(\Omega, b)db\right),$$

it yields

$$\nabla f^{\text{obj}}(\mathbf{x}) = \int_{\Omega\in S^d} w(\Omega, \Omega\cdot\mathbf{x})\Omega d\Omega.$$

Take w as the half Fourier transform of another unknown function z, that is $w(\Omega, a) = \int_{\mu>0} e^{i\mu a}z(\Omega, \mu)d\mu$. One gets

$$\nabla f^{\text{obj}}(\mathbf{x}) = \int_{\Omega\in S^d}\int_{\mu>0} e^{i\mu\Omega\cdot\mathbf{x}}z(\Omega, \mu)\Omega d\mu d\Omega.$$

Make the change of variable $\mathbf{k} = \Omega\mu$ that is $\mu = |\mathbf{k}|$ and $\Omega = \frac{\mathbf{k}}{|\mathbf{k}|}$. The equivalence of the measures writes $d\mathbf{k} = \mu^{d-1} d\mu d\Omega$. One gets

$$\nabla f^{\text{obj}}(\mathbf{x}) = \int_{\mathbf{k}\in\mathbb{R}^d} \left(\frac{z(\Omega,\mu)}{\mu^d} \right) \mathbf{k} e^{i\mathbf{k}\cdot\mathbf{x}} d\mathbf{k}.$$

The Fourier representation writes $f^{\text{obj}}(\mathbf{x}) = \frac{1}{2\pi} \int_{\mathbf{k}\in\mathbb{R}^d} \widehat{f^{\text{obj}}}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}} d\mathbf{k}$, so

$$\nabla f^{\text{obj}}(\mathbf{x}) = \frac{i}{2\pi} \int_{\mathbf{k}\in\mathbb{R}^d} \widehat{f}(\mathbf{k}) \mathbf{k} e^{i\mathbf{k}\cdot\mathbf{x}} d\mathbf{k}.$$

Note that $\widehat{f^{\text{obj}}}(\mathbf{k}) = \int_{\mathbf{x}\in\mathbb{R}^d} f^{\text{obj}}(\mathbf{x}) e^{i\mathbf{k}\cdot\mathbf{x}} d\mathbf{x}$ and $f^{\text{obj}} \in H_0^m(\mathbb{R}^d)$ show that for all $q \geqslant 1$, then

$$|\mathbf{k}|^m \widehat{f^{\text{obj}}}(\mathbf{k}) \in W^{q,\infty}(\mathbb{R}^d) \text{ as a function of } \mathbf{k}.$$

All this shows that a good choice of z is

$$z(\Omega,\mu) = \frac{i}{2\pi} \mu^d \widehat{f}^{\text{obj}}(\mu\Omega).$$

For $m \geqslant d+2$, this is a well defined function which decreases like $O(\mu^{-2})$ for large $\mu > 0$. Then w is well defined and bounded in $L^\infty(\mathbb{R})$ with respect to a, for all Ω. With this formula, one gets w.

By d integration by parts on μ, there exists $C > 0$ such that

$$|w(\Omega,a)| \leqslant C a^{-d}, \quad a > 0.$$

It is enough to to be sure the formula (41) is integrable for $d \geqslant 2$.

□

Then as before, by Riemann discretization procedure and regularization of the Heaviside functions, it is possible to obtain in any dimensions formulas similar to the previous ones in dimension one.

3.1.3 The proof of Cybenko

The original proof is based on theoretical arguments.

Definition 3.11. *Assume a sigmoid has the following property: for any non negative measure $d\mu$,*

$$\int_{I_d} \sigma(\mathbf{y}\cdot\mathbf{x} + \theta) d\mu(\mathbf{x}) = 0, \quad \forall (\mathbf{y},\theta) \in \mathbb{R}^d \times \mathbb{R},$$

implies that $\mu = 0$. We say the sigmoid is discriminatory.

Theorem 3.12. *Assume that σ is discriminatory. Then Theorem 3.1 holds.*

Proof. Take the set S made of all functions of the form (34). It is a linear sub-space of $C^0(I_d)$. Take its closure $\overline{S}$ in $C^0(I_d)$ for the norm of the maximum. If $\overline{S} = C^0(I_d)$, the proof is ended.

Assume on the contrary that $\overline{S} \neq C^0(I_d)$. By the Hahn-Banach Theorem [3], there exists a bounded linear form $L \neq 0$ on $C^0(I_d)$ such that $L(\overline{S}) = 0$.

By the Riesz Theorem, there exists a non negative measure $\mu \neq 0$ such that

$$L(h) = \int_{I_d} h(x) d\mu(x), \qquad h \in C^0(I_d).$$

Taking $h(x) = \sigma(\mathbf{y} \cdot \mathbf{x} + \theta)$ for all $(\mathbf{y}, \theta) \in \mathbb{R}^d \times \mathbb{R}$ yields a contradiction. So the proof. □

It remains to prove the discriminatory property.

Lemma 3.13. *Any continuous monotone increasing sigmoid is discriminatory.*

Proof. Set $\sigma^{\lambda}_{\mathbf{y},\theta}(\mathbf{x}) = \sigma\left(\lambda(\mathbf{y} \cdot \mathbf{x} + \theta) + \varphi\right)$, $\Pi_{\mathbf{y},\theta} = \{\mathbf{x} \mid \mathbf{y} \cdot \mathbf{x} + \theta = 0\}$ and $H_{\mathbf{y},\theta} = \{\mathbf{x} \mid \mathbf{y} \cdot \mathbf{x} + \theta > 0\}$. One has $0 = \int_{I_d} \sigma^{\lambda}_{\mathbf{y},\theta}(\mathbf{x}) d\mu(\mathbf{x})$. Passing to the limit $\lambda \to +\infty$ yields

$$0 = \sigma(\varphi)\mu\left(\Pi_{\mathbf{y},\theta}\right) + \mu\left(H_{\mathbf{y},\theta}\right).$$

So $\mu\left(\Pi_{\mathbf{y},\theta}\right) = \mu\left(H_{\mathbf{y},\theta}\right) = 0$.

For a given $\mathbf{y} \in \mathbb{R}^d$, set

$$F_{\mathbf{y}}(h) = \int_{I_d} h(\mathbf{y} \cdot \mathbf{x}) d\mu(\mathbf{x}).$$

From $\mu\left(H_{\mathbf{y},\theta}\right) = 0$, then $F_{\mathbf{y}}(h) = 0$ is h is the indicatrix function of an interval. By density, $F_{\mathbf{y}}(h) = 0$ for all $\mathbf{y} \in \mathbb{R}^d$. Then $F_{\mathbf{y}} = 0$.

Then

$$\int_{I_d} e^{i\mathbf{k}\cdot\mathbf{x}} d\mu(x) = 0 \quad \forall \mathbf{k} \in \mathbb{R}^d.$$

That is the Fourier transform of the measure is zero. So $\mu = 0$ as well [13][p. 176]. □

3.2 Connection with Finite Elements

Now we state a known result which shows that recursivity helps to reconstruct basic finite elements shape functions, and also it will make visible the connection with basic methods in the theory of numerical approximation of solutions of PDEs.

3.2.1 1D piecewise affine finite elements

Take a non uniform mesh in dimension one

$$\cdots < x_{j-1} < x_j < x_{j+1} < \ldots, \qquad \text{with } \lim_{j \to \pm\infty} = \pm\infty.$$

We note $h_{j+\frac{1}{2}} = x_{j+1} - x_j$.

Definition 3.14. *The adaptative Finite Element [1] P_1 function φ_j is*

$$\varphi_j(x) = \begin{cases} 0 & \text{for } x \leqslant x_{j-1}, \\ \frac{x - x_{j-1}}{h_{j-\frac{1}{2}}} & \text{for } x_{j-1} \leqslant x \leqslant x_j \\ \frac{x_{j+1} - x}{h_{j+\frac{1}{2}}} & \text{for } x_j \leqslant x \leqslant x_{j+1} \\ 0 & \text{for } x_{j+1} \leqslant x. \end{cases}$$

Lemma 3.15. *One has the three different representations*

$$\begin{cases} \varphi_j(x) = R\left(\frac{1}{h_{j-\frac{1}{2}}}R(x-x_{j-1}) - \left(\frac{1}{h_{j-\frac{1}{2}}} + \frac{1}{h_{j+\frac{1}{2}}}\right)R(x-x_j)\right), \\ \varphi_j(x) = R\left(\frac{1}{h_{j+\frac{1}{2}}}R(x_{j+1}-x) - \left(\frac{1}{h_{j-\frac{1}{2}}} + \frac{1}{h_{j+\frac{1}{2}}}\right)R(x-x_j)\right), \\ \varphi_j(x) = T\left(T\left(\frac{x-x_{j-1}}{h_{j-\frac{1}{2}}}\right) + T\left(\frac{x_{j+1}-x}{h_{j+\frac{1}{2}}}\right) - 1\right). \end{cases}$$

The second formula is the symmetrized version of the first one. The third representation is symmetric left-right.

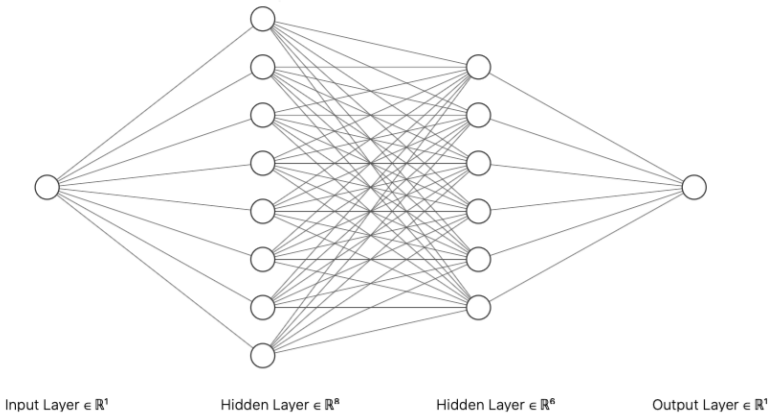

Figure 4: Graph of the network which implements (42). The first hidden layer is two more neurons than the second hidden layer.

One can approximate a given function objective function $f^{\text{obj}} \in C^0(\mathbb{R})$ with the well known Finite element interpolation formula

$$f(x) = \sum_{j\in\mathbb{Z}} f^{\text{obj}}(x_j)\varphi_j(x). \tag{42}$$

Lemma 3.16. *One has $f(x_j) = f^{\text{obj}}(x_j)$ for all $j \in \mathbb{Z}$ and*

$$\|f^{\text{obj}} - f\|_{L^\infty(\mathbb{R})} \leqslant \frac{1}{8}\sup_{j\in\mathbb{Z}}\left(\|(f^{\text{obj}})''\|_{L^\infty(x_j,x_{j+1})}h_{j+\frac{1}{2}}^2\right).$$

Proof. The function f is the P_1 Lagrange interpolation of f^{obj}, so the first part of the claim is immediate. The second part is super classical for Lagrange interpolation. For a given $x \in (x_j, x_{j+1})$, take

$$\mu = \frac{f^{\text{obj}}(x) - f(x)}{\frac{(x-x_j)(x-x_{j+1})}{2}}.$$

and consider

$$g(y) = f^{\text{obj}}(y) - f(y) - \mu\frac{(y-x_j)(y-x_{j+1})}{2}.$$

By construction $g(x_j) = g(x_{j+1}) = g(x) = 0$ and $g''(y) = (f^{\rm obj})''(y) - \mu$. Since g vanishes at three different points, the Role theorem used two times yields that there exists $y \in (x_j, x_{j+1})$ such that $g''(y) = 0$. So $\mu = g''(y)$ from which the result is deduced using $f^{\rm obj}(x) - f(x) = \mu\frac{(x-x_j)(x-x_{j+1})}{2}$. □

To reach a given accuracy ε, one needs to satisfy $\frac{1}{8}\left(\|(f^{\rm obj})''\|_{L^\infty(x_j,x_{j+1})}h^2_{j+\frac{1}{2}}\right) \leqslant \varepsilon$, for all $j \in \mathbb{Z}$, that is

$$h_{j+\frac{1}{2}} \leqslant \sqrt{\frac{8\varepsilon}{\|(f^{\rm obj})''\|_{L^\infty(x_j,x_{j+1})}}}, \quad \forall j \in \mathbb{Z}.$$

That is the more flat the function locally is, the less is $\|(f^{\rm obj})''\|_{L^\infty(x_j,x_{j+1})}$ is, the more one can take large distance between points. This is adaptative numerical approximation. The hope is that sofwares can capture such a phenomenon.

Actually, it is possible to obtain the same result with one less layer, still with ReLU activation functions.

Lemma 3.17. *One has*

$$\begin{cases} f(x) = \sum_{j\in\mathbb{Z}} \alpha_j R(x - x_j), \quad \alpha_j = \frac{f(x_{j+1})-f(x_j)}{h_{j+\frac{1}{2}}} - \frac{f(x_j)-f(x_{j-1})}{h_{j-\frac{1}{2}}}, \\ f(x) = \sum_{j\in\mathbb{Z}} \beta_j T\left(\frac{x-x_j}{h_{j+\frac{1}{2}}}\right), \quad \beta_j = f(x_{j+1}) - f(x_j). \end{cases}$$

Proof. Evident. Many other representations are possible. □

3.2.2 ReLU function and power of depth

This discussion is based on a recent remarkable result by Yarosky [16], then exploited in [6] and other works. It is based of the class of Takagi functions. It starts with a formula in relation with the ReLU function. Essentially, one shows that ReLU is a universal multiplier with spectral accuracy.

Let $g : [0,1] \longrightarrow [0,1]$ be the hat function in dimension one

$$g(x) = 1 - |1 - 2x| = 2R(x) - 4R\left(x - \frac{1}{2}\right), \quad 0 \leqslant x \leqslant 1.$$

For $0 \leqslant x \leqslant 1$, w notice that $g(x/2) = x$ and $g(1 - x) = g(x)$.

Definition 3.18. *By iteration one defines the family of saw-tooth functions: $g_{n+1} = gog_n$ with $g_1 = g$ (or with $g_0(x) = x$).*

Lemma 3.19. *For $n \geqslant 1$, g_n is such that*

$$g_n(x) = 2^n x \text{ for } 0 \leqslant x \leqslant \frac{1}{2^n} \tag{43}$$

and symmetric with respect to $\frac{1}{2^n}$

$$g_n(x) = g_n\left(\frac{1}{2^{n-1}} - x\right) \text{ for } \frac{1}{2^n} \leqslant x \leqslant \frac{1}{2^{n-1}}. \tag{44}$$

For $n \geqslant 2$, g_n is $\frac{1}{2^{n-1}}$-periodic on $[0,1]$

$$g_n\left(x+\frac{1}{2^{n-1}}\right) = g_n(x) \text{ for } 0 \leqslant x \leqslant 1-\frac{1}{2^{n-1}}. \tag{45}$$

Proof. By definition, one has $g_1(x) = 2x$ for $0 \leqslant x \leqslant \frac{1}{2}$. Then (43) holds by iteration on n. Take $\frac{1}{2^n} \leqslant x \leqslant \frac{1}{2^{n-1}}$, then

$$g_n(x) = g\left(g_{n-1}(x)\right) = g\left(2^{n-1}x\right) = g\left(1-2^{n-1}x\right) = g\left(g_{n-1}\left(\frac{1}{2^{n-1}}-x\right)\right) = g_n\left(\frac{1}{2^{n-1}}-x\right)$$

that is (44).

Let us show (45) first for $n = 2$. For $0 \leqslant x \leqslant \frac{1}{2}$, one has

$$g_2\left(x+\frac{1}{2}\right) = g_2\left(1-\left(x+\frac{1}{2}\right)\right) = g_2\left(\frac{1}{2}-x\right) = g_2(x) \qquad \text{(using (44))}.$$

For $n \geqslant 2$ and $0 \leqslant x \leqslant \frac{1}{2^{n-1}}$, one has

$$g_n\left(x+\frac{1}{2^{n-1}}\right) = g_2\left(g_{n-2}\left(x+\frac{1}{2^{n-1}}\right)\right) = g_2\left(2^{n-2}\left(x+\frac{1}{2^{n-1}}\right)\right)$$

$$= g_2\left(2^{n-2}x+\frac{1}{2}\right) = g_2\left(2^{n-2}x\right) = g_2\left(g_{n-2}(x)\right) = g_n(x)$$

which is (45). □

Lemma 3.20. *Let $0 \leqslant x \leqslant 1$. One has $x^2 = x - \sum_{n\geqslant 1} \frac{1}{4^n} g_n(x)$.*

Proof. The series is convergent. Set

$$H(x) = x - \sum_{n\geqslant 1} \frac{1}{4^n} g_n(x) = \sum_{n\geqslant 1} \frac{1}{4^n} h_n(x), \quad h_n(x) = \left(2^n x - g_n(x)\right).$$

By construction $h_n(x/2) = h_{n-1}(x)$ and $h_1(x/2) = 0$, so

$$H\left(\frac{x}{2}\right) = \frac{1}{4}H(x). \tag{46}$$

One also has

$$H(1-x) = 1 - x - \sum_{n\geqslant 1} \frac{1}{4^n} g_n(1-x) = 1 - x - \sum_{n\geqslant 1} \frac{1}{4^n} g_n(x) = 1 - 2x + H(x). \tag{47}$$

It shows (48), that is $H(x) = x^2$ for $x \in \left\{\frac{k}{2^n} : n \in \mathbb{N},\ k = 0, \ldots, 2^n\right\}$ which is a dense subset of the interval $[0,1]$. By direct calculation, one has that $\|g_n'\|_{L^\infty} = 2^n$ so H is Lipschitz

$$|H(x) - H(y)| = |x-y| + \sum_{n\geqslant 1} \frac{1}{4^n} 2^n |x-y| = 2|x-y|,$$

and it is continuous. Therefore $H(x) = x^2$ for all x in the interval $[0,1]$. The claim is proved. □

Lemma 3.21. *One has*

$$H\left(\frac{k}{2^n}\right) = \left(\frac{k}{2^n}\right)^2, \qquad 0 \leqslant k \leqslant 2^n, \quad n \in \mathbb{N}. \tag{48}$$

Proof. By definition, $H(1) = 1$ and $H(0) = 0$ so (48) holds for $n = 0$. For $n = 1$, take (46) with $x = 1$. Assume now (48) for $n \geqslant 1$. Then (46) shows

$$H\left(\frac{k}{2^{n+1}}\right) = \frac{1}{4}\left(\frac{k}{2^n}\right)^2 = \left(\frac{k}{2^{n+1}}\right)^2, \qquad 0 \leqslant k \leqslant 2^n.$$

Next take $2^n < k \leqslant 2^{n+1}$ and note $k' = 2^{n+1} - k$ such that $0 \leqslant k' \leqslant 2^n$. Then, with (47), one has

$$H\left(\frac{k}{2^{n+1}}\right) = H\left(1 - \frac{k'}{2^{n+1}}\right) = 1 - 2\frac{k'}{2^{n+1}} + H\left(\frac{k'}{2^{n+1}}\right) = 1 - 2\frac{k'}{2^{n+1}} + \left(\frac{k'}{2^{n+1}}\right)^2 = \left(\frac{k}{2^{n+1}}\right)^2$$

and which shows (48) for $n + 1$. Iteration on n ends the proof. □

Lemma 3.22. *One has fast convergence of the series*

$$\left\| x^2 - \left(x - \sum_{n=1}^{p} \frac{1}{4^n} g_n(x) \right) \right\|_{L^\infty(0,1} \leqslant \frac{1}{3 \times 4^p}. \tag{49}$$

Proof. Indeed $\left\| \sum_{n=p+1}^{\infty} \frac{1}{4^n} g_n(x) \right\|_{L^\infty(0,1} \leqslant \sum_{n=p+1}^{\infty} \frac{1}{4^n} = \frac{1}{3 \times 4^p}$. □

Next we show how to define a neural network which implements

$$f(x) = x - \sum_{n \geqslant 1}^{p} \frac{1}{4^n} g_n(x). \tag{50}$$

Since $f(x)$ is an approximation of x^2, it is an approximation of the multiplication of x by itself.

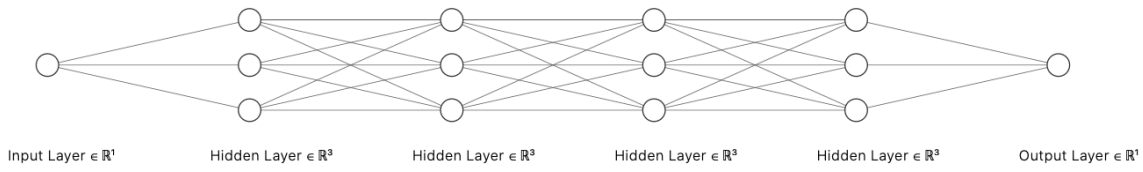

Figure 5: Graph structure of the Takagi formula for $p = 4$.

Let is firstly construct two networks which realize f. We see on Figure 5 a network for $N = 4$. It is made of 4 hidden dense layers of width equal to 3, plus the input and output layer. The output layer uses a linear activation function. The implementation of f is made with p hidden layers with formula (14). The two top neurons of each layer on the figure are here to calculate the

g_n for increasing values of n. All the bottom neurons make what is called a collation channel, which calculates the sum. Using the notations of (14), the network is defined by

$$W_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad b_0 = \begin{pmatrix} 0 \\ -\frac{1}{2} \\ 0 \end{pmatrix}, \quad g = R,$$

$$W_r = \begin{pmatrix} 2 & -4 & 0 \\ 2 & -4 & 0 \\ -4^{-r} & 0 & 1 \end{pmatrix}, \quad b_r = \begin{pmatrix} -2 \\ 4 \\ 4^{-r} \end{pmatrix}, \quad g = R, \quad 1 \leqslant r \leqslant p-1,$$

$$W_p = \begin{pmatrix} -2 \times 4^{-p}, & -4 \times 4^{-p}, & 1 \end{pmatrix}, \quad b_p = \begin{pmatrix} 0 \end{pmatrix}, \quad g = \text{linear}.$$

Exercise 3.23. *Prove a key property: the ReLU is not activated on the collation channel, because all results are non positive.*

Lemma 3.24. *The function f can be approximated in the maximal norm on a neural network with p hidden dense layers with 3 neurons, with error $\varepsilon = O(4^{-p})$. The scaling of the numerical cost with respect the error is*

$$\text{Cost} = O(\log 1/\varepsilon).$$

Proof. The scaling $\varepsilon = O(4^{-p})$ comes from (49-50). For large number p of hidden layers, the cost is proportional is $\text{Cost} = O(3p)$. But $p = O(\log 1/\varepsilon)$. So the claim. □

The exact values of the matrices of weights and offsets is of course not unique. This is already the consequence of Lemma 2.9, considering permutations. Changing the activation function and the values is another possibility. For example, the network below keep the same result. Using the notations of (14) and the ReLU with threshold $a = 1$, the network is defined by

$$\overline{W}_0 = \begin{pmatrix} 2 \\ -2 \\ 1 \end{pmatrix}, \quad b_0 = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}$$

$$\overline{W}_r = \begin{pmatrix} 2 & 2 & 0 \\ -2 & -2 & 0 \\ -4^{-r} & -4^{-r} & 1 \end{pmatrix}, \quad b_r = \begin{pmatrix} -2 \\ 4 \\ 4^{-r} \end{pmatrix}, \quad 1 \leqslant r \leqslant p-1,$$

$$\overline{W}_p = \begin{pmatrix} -4^{-p}, & -4^{-p}, & 1 \end{pmatrix}, \quad b_p = \begin{pmatrix} 4^{-p} \end{pmatrix}, \quad g = \text{linear}.$$

Exercise 3.25. *Check it gives the same result.*

There are some reasons to think the second implementation is better than the first one, because the sensibility to variations with respect of x or to the weights and offsets is less.

Lemma 3.26. *Neural network approximations of $x \mapsto x^n$ are possible by various replications of a first one for $x \mapsto x^2$. The same for neural network approximations of $(x, y) \mapsto xy$.*

Proof. We detail the proof for $n = 3$ on the basis of the polarization formula

$$x^3 = \frac{1}{4}(x + x^2)^2 - \frac{1}{4}(x - x^2)^2. \tag{51}$$

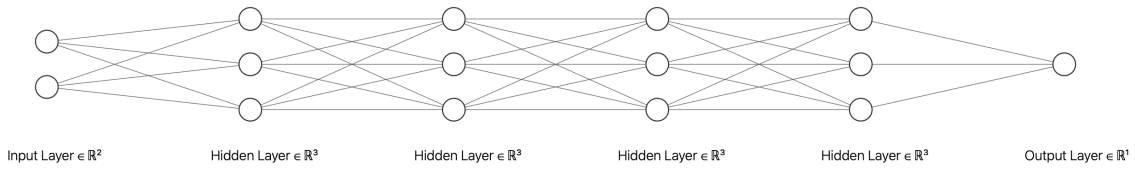

Figure 6: Graph structure of a universal multiplier $(x, y) \mapsto xy$ with $m = 2$, $p = 4$ and $n = 1$.

Add a trivial collocation channel to the structure of Figure 5. Then one can construct a new function which realizes an approximation of $f_1^{\text{obj}} : x \mapsto \begin{pmatrix} x^2 \\ x \end{pmatrix}$. A second function is

$$\begin{array}{rccc} f_2^{\text{obj}} : & \mathbb{R}^2 & \to & \mathbb{R}^3 \\ & \begin{pmatrix} x \\ y \end{pmatrix} & \to & \begin{pmatrix} x \\ y \\ \frac{1}{2}(x+y) \end{pmatrix}. \end{array}$$

By definition $f_2^{\text{obj}} o f_1^{\text{obj}}(x) = \left(x^2, x, \frac{1}{2}(x^2 + x)\right)^t$. Our last function is $f_3^{\text{obj}}(x) = x^2$, defined by convention between $\mathbb{R}^3 \to \mathbb{R}^2$. Finally consider $f_4^{\text{obj}}(x, y, z) = -\frac{1}{2}x^2 - \frac{1}{2}y + 2z^2$. One checks that

$$f_4^{\text{obj}} \circ f_3^{\text{obj}} \circ f_2^{\text{obj}} \circ f_1^{\text{obj}}(x) = x^3.$$

With convenient neural network approximations, the proof is ended. □

Theorem 3.27 (Yarotsky)**.** *There exists a*

Proof. The proof [17] is based on the error formula of Lemma 3.24 for the function $x \mapsto x^2$, on Lemma 3.26 to obtain similar results for all monomials such as $x^n y^m$, and on explicit manipulations for functions in the considered space which can be approximated by polynomials (using for example variants of the Weierstrass Theorem). □

3.2.3 The Takagi function

The Takagi function [15] illustrates a fascinating property of neural networks, which is the ability of approximating very irregular function. This feature is totally counterintuitive with respect to the standard approximation theory of smooth functions.

Definition 3.28. *The Takagi function* $\text{Tak} \in C^0[0, 1]$ *is*

$$\text{Tak}(x) = \sum_{n \geqslant 1} \frac{1}{2^n} g_n(x).$$

Lemma 3.29. *One has the symmetry property* $\text{Tak}(1 - x) = \text{Tak}(x)$ *and the self similar property*

$$\text{Tak}\left(\frac{x}{2}\right) = \frac{1}{2}x + \frac{1}{2}\text{Tak}(x). \tag{52}$$

Proof. Indeed

$$\mathrm{Tak}\left(\frac{x}{2}\right) = \frac{1}{2}g\left(\frac{x}{2}\right) + \sum_{n\geqslant 2}\frac{1}{2^n}g_n(x/2) = \frac{1}{2}x + \sum_{n\geqslant 2}\frac{1}{2^n}g_{n-1}(x) = \frac{1}{2}x + \frac{1}{2}\mathrm{Tak}(x).$$

□

Lemma 3.30. *Check that* Tak *can be approximated in the maximal norm on a neural network with p hidden dense layers with 3 neurons, with error $O(2^{-p})$.*

Proof. Adapt the proof of Lemma 3.24. □

It appears that the Takagi function is nowhere differentiable. For the simplicity of the proof, we specify the result with respect to quadratic norm.

Theorem 3.31. *Whatever $0 \leqslant a < b \leqslant 1$, then* $\mathrm{Tak} \notin H^1(a,b)$.

Proof. They are many proofs, all of them having their own interest. This one is by contradiction and self similarity.

First we treat $a = 0$ and $b = 1$. Assume by contradiction that Tak is in $H^1(0,1)$. One has by differentiation $\mathrm{Tak}'\left(\frac{x}{2}\right) = 1 + \mathrm{Tak}'(x)$. So

$$\int_0^1 \mathrm{Tak}'(x)^2 dx = \int_0^1 \left(\mathrm{Tak}'\left(\frac{x}{2}\right) - 1\right)^2 dx = \int_0^1 \mathrm{Tak}'\left(\frac{x}{2}\right)^2 dx + 1 = 2\int_0^{\frac{1}{2}} \mathrm{Tak}'\left(y\right)^2 dy + 1.$$

So $\int_0^1 \mathrm{Tak}'\left(x\right)^2 dx = \int_0^1 \mathrm{Tak}'\left(x\right)^2 dx + 1$ which is a contradiction.

Second we consider the general case. Take a small enough interval $J = \left[\frac{k}{2^{m-1}}, \frac{k+1}{2^{m-1}}\right] \subset [a,b]$ for $0 \leqslant k \leqslant 2^{m-1} - 1$ where m is large enough. The self-similar properties of the g_n's yield that

$$\mathrm{Tak}\left(\frac{k}{2^{m-1}} + x\right) = \sum_{n=1}^{m-1}\frac{1}{2^n}g_n(x) + \frac{1}{2^{m-1}}\mathrm{Tak}(2^{m-1}x).$$

The functions g_n's are all in $H^1(0,1)$ (general result which comes from Finite Element Method theory). When x spans the small interval $J \subset [0,1]$, then $2^{m-1}x - k$ spans the entire interval $[0,1]$. Therefore the first point shows that $\mathrm{Tak} \notin H^1\left(\frac{k}{2^{m-1}}, \frac{k+1}{2^{m-1}}\right)$ and the proof is ended. □

4 Fixed point approach to NN

It has been observed recently in [17, 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 [17], this generalization is the basis of a general theorem of approximation of functions by Neural Network architectures, see also [13]. 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.

4.1 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 $m \geqslant 1$ 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

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

The set of continuous piecewise linear functions is

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

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 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 4.8. 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, 17, 5, 6, 13].

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 4.1. *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*

$$H(x) = e_0(x) + \sum_{i=1}^{r} \beta_i H(e_i(x)), \quad x \in I, \tag{53}$$

with the contraction condition

$$K < 1, \qquad K = \sum_{i=1}^{r} |\beta_i|. \tag{54}$$

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 (53) 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, 17, 5]

$$H_1(x) = \frac{1}{4}g(x) + \frac{1}{4}H_1(g(x)) \tag{55}$$

where g is the hat function (normalized finite element function): $g(x) = 2x$ for $0 \leqslant x \leqslant \frac{1}{2}$ and $g(x) = 2(1-x)$ for $\frac{1}{2} \leqslant x \leqslant 1$. 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

$$H_1(x) = e_0(x) + \frac{1}{4}H_1(e_1(x)) + \frac{1}{4}H_1(e_2(x)), \quad e_0(x) = \frac{1}{4}(g(x) - 1) \tag{56}$$

where the contraction property (54) 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 $\deg(H) \geqslant 2$. 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 $|\mu_i| \leqslant 1$. So $1 \leqslant \sum |\beta_i||\mu_i|^n \leqslant \sum |\beta_i|$. It means the contraction condition (54) is not satisfied and Problem 4.1 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 4.2. *Let $H \in P^n$. There exists a threshold value $m_*(H) \geqslant 2$ such that the functional equation (53) has a solution with the contraction property (54) for all $m \geqslant m_*(H) \iff h \leqslant 1/m_*(H)$.*

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 4.6 in the general case. They can also be taken as in Lemma 4.5 if $H(x) = x^n$ is a monomial.

For $n \geqslant 2$, 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 4.3. *For all subintervals $0 \leqslant j \leqslant m-1$, find triples $(a_i, b_i, \gamma_i) \in I \times I \times \mathbb{R}$ $(1 \leqslant i \leqslant s)$ such that $b_i - a_i \neq 0$ for all i and*

$$p(x_j + hy) = \sum_{i=1}^{s} \gamma_i p(a_i + (b_i - a_i)y), \qquad y \in I. \tag{57}$$

Lemma 4.4. *The equation (53) is equivalent to the equation (57).*

Proof. The proof is in two parts.
(53) $\Rightarrow$ (57): 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 (53) 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 (57) where $x = x_j + hy$ and $e_i(x) = a_i + (b_i - a_i)y$. (57) $\Rightarrow$ (53): rewrite the discrete quantities in (57) 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

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

Define also

$$\beta_{i,j} = \frac{h^2}{(b_{i,j} - a_{i,j})^2}\gamma_{i,j}, \qquad \text{where } b_{i,j} - a_{i,j} \neq 0. \tag{58}$$

Consider the function

$$e_0(x) = H(x) - \sum_j \sum_i \beta_{i,j} H(e_{i,j}(x)). \tag{59}$$

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 $y \mapsto p(a_{i,j} + (b_{i,j} - a_{i,j})y)$, $1 \leqslant i \leqslant s$, generate a complete system in P^{n-2}, then the equation (57) has a solution. That is why we will consider from now on that

$$s = \dim(P^{n-2}) = n - 1. \tag{60}$$

Next, by differentiation, the equation (57) in $[x_j, x_{j+1}]$ is equivalent to the square linear system

$$M_j X_j = b_j, \qquad 0 \leqslant j \leqslant m - 1. \tag{61}$$

The square matrix is

$$M_j = \begin{pmatrix} p(a_{1,j}) & p(a_{2,j}) & \dots & p(a_{n-1,j}) \\ c_{1,j}p'(a_{1,j}) & c_{2,j}p'(a_{2,j}) & \dots & c_{n-1,j}p'(a_{n-1,j}) \\ \dots & \dots & \dots & \dots \\ c_{1,j}^{n-2}p^{(n-2)}(a_{1,j}) & c_{2,j}^{n-2}p^{(n-2)}(a_{2,j}) & \dots & c_{n-1,j}^{n-2}p^{(n-2)}(a_{n-1,j}) \end{pmatrix} \in \mathcal{M}_{n-1}(\mathbb{R}) \tag{62}$$

where we note $c_{i,j} = b_{i,j} - a_{i,j}$. The unknown of the linear system is $X_j = \left(\gamma_{1,j}, \gamma_{2,j}, \dots, \gamma_{n-1,j}\right)^T \in \mathbb{R}^{n-1}$. The right hand side of the linear system is $b_j = \left(p(x_j), hp'(x_j), \dots, h^{n-1}p^{(n-1)}(x_j)\right)^T \in \mathbb{R}^{n-1}$ which is bounded $\|b_j\|_\infty \leqslant C$ 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 4.5. *Take $p(x) = x^{n-2}$ with $n - 2 \geqslant 0$. 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 $1 \leqslant i \neq k \leqslant n - 2$.*

Proof. The matrix is $M = \left(\frac{n!}{(n-t)!}(b_i - a_i)^t a_i^{n-t}\right)_{1\leqslant t+1, i\leqslant n-1}$. By assumption $b_i - a_i \neq 0$ for all i, so M is similar to $N = \left(\left(\frac{a_i}{(b_i-a_i)}\right)^{n-t}\right)_{1\leqslant t+1, i\leqslant n-1}$. 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 4.6. *Take $p \in P^{n-2}$ with $deg(p) = n-2 \geqslant 0$. 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 $1 \leqslant i \neq k \leqslant n-2$. Then the matrix M is non singular.*

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

Lemma 4.7. *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 4.2. If $n = 0$ or $n = 1$ the result is trivial, so we consider $n-2 \geqslant 0$. If $H(x) = x^n$ is a monomial function, one can takes the first set of basis functions given by Lemma 4.5 because the matrices are non singular. Unfortunately, this simple choice is not always possible as shown by Lemma 4.7. So to cover the case of general polynomials $H \in P^n$, we continue with basis functions satisfying Lemma 4.6.

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

The matrix $M_j = M$ being non singular, then the system (61) has a solution $X_j = (\gamma_{1,j}, \ldots, \gamma_{n-1,j})$ such that $\|X_j\|_\infty \leqslant \|M^{-1}\|_\infty \|b_j\|_\infty \leqslant C$ uniformly with respect to the index of the subinterval j. So the representation (59) of H holds for $x \in I$.

The constant is

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

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

Lemma 4.8. *Under the contraction condition (54), one has the bounds $\|H\|_{L^\infty(I)} \leqslant \frac{1}{1-K}\|e_0\|_{L^\infty(I)}$ and $\|\sum_{i\geqslant 1} \beta_i H \circ e_i\|_{L^\infty(I)} \leqslant \frac{K}{1-K}\|e_0\|_{L^\infty(I)}$.*

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

$$\begin{array}{rccc} \mathcal{H}: & L^\infty(I) & \longrightarrow & L^\infty(I) \\ & G & \longmapsto & \sum_{i\geqslant 1} \beta_i G \circ e_i. \end{array} \tag{63}$$

Then $\|\mathcal{H}\|_{\mathcal{L}(L^\infty(I))} \leqslant K < 1$, 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. □

4.2 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, 17, 8]. These algorithms provide reference solutions based on the standard fixed point method where the iteration index is $k = 0, 1, \ldots$

$$\begin{cases} H_0 = 0, \\ H_{k+1} = e_0 + \displaystyle\sum_{1 \leqslant i \leqslant r} \beta_i H_k \circ e_i. \end{cases} \tag{64}$$

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

$$H_k = e_0 + \sum_{p=1}^{k-1} \left(\sum_{1 \leqslant i_1, \ldots, i_p \leqslant r} \left(\beta_{i_1} \ldots \beta_{i_p}\right) e_0 \circ e_{i_1} \circ \cdots \circ e_{i_p} \right), \quad k \geqslant 1. \tag{65}$$

Lemma 4.8 and the contractive operator (63) yield a standard convergence property with exponential rate in $L^\infty(I)$

$$\|H_k - H\|_{L^\infty(I)} \leqslant K^k \|H\|_{L^\infty(I)}, \qquad K = \sum_{1 \leqslant i \leqslant r} |\beta_i| < 1. \tag{66}$$

4.2.1 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 [7]

$$T(x) = \max(0, \min(x, 1)), \qquad \text{with } T(x) = \min(R(x), 1) = R(x) - R(x-1). \tag{67}$$

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 (67), 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 $\mathcal{E}$

$$\begin{aligned} \mathcal{E} = \big\{ e \in C^0(I) : \quad & \text{there exists } 0 \leqslant \alpha < \beta \leqslant 1 \text{ and } 0 \leqslant a \leqslant b \leqslant 1 \text{ such that} \\ & e(x) = a \text{ for } 0 \leqslant x \leqslant \alpha, \\ & e(x) = a + (b-a)\tfrac{x-\alpha}{\beta-\alpha} \text{ for } \alpha \leqslant x \leqslant \beta, \\ & e(x) = b \text{ for } \beta \leqslant x \leqslant 1 \big\}. \end{aligned} \tag{68}$$

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 $L_1 \circ L_2 = L_3$.

[7] 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 4.9. *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 (68), $e(x) = a + (b-a)T\left((x-\alpha)/(\beta-\alpha)\right)$. □

Lemma 4.10. *The series (65) 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 4.2.1. These operations are already sufficient to implement the iterations (64).

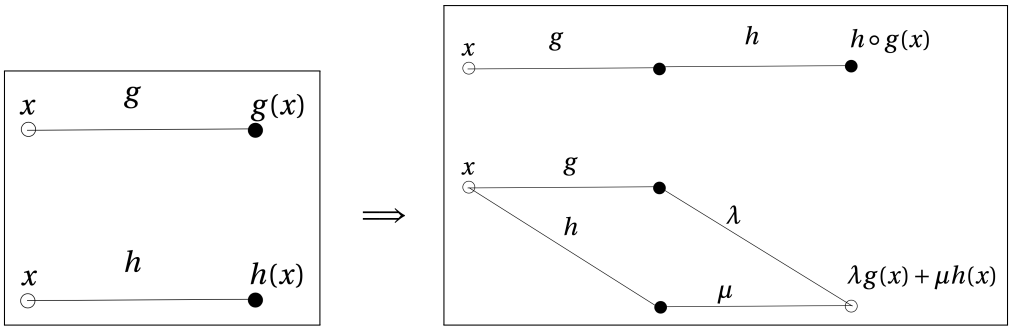

Figure 7: Sketch in a Neural Network of the composition $h \circ g$ and of the linear combination $\lambda g + \mu h$ of 2 functions g and h already implemented as Neural Networks. Empty bullets correspond either to the input or to pure linear combination. Black bullets show that an activation function R, T or another one is used.

Now consider (65). 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, 17, 8] for the polynomial $x \mapsto x - x^2$.

4.2.2 Accuracy

Considering (66), 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 (65) 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 $\mathbf{k} = \mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots$ in the series (65). 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 (65): 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 (66), so there is a balance between the cost and the accuracy.

4.2.3 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.

4.2.4 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 $G(t) \in L^\infty(I)$

$$\left\{ \begin{array}{l} G(t) = 0, \\ G'(t) = e_0 + \displaystyle\sum_{1 \leqslant i \leqslant r} \beta_i G(t) \circ e_i - G(t). \end{array} \right. \tag{69}$$

One has the bound $\|G(t) - H\|_{L^\infty(I)} \leqslant e^{-(1-K)t} \|H\|_{L^\infty(I)}$. A possible splitting technique is based on decreasing time steps $\Delta t_1 \geqslant \cdots \geqslant \Delta t_k \geqslant \Delta t_{k+1}$ 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 $t_k \leqslant t \leqslant t_{k+1} = t_k + \Delta t_{k+1}$, one may consider the series of ODEs

$$\left\{ \begin{array}{ll} G_0'(t) = e_0 - (1 - \sum_{i=1}^r |\beta_i|) G(t), & 0 \leqslant t \leqslant \Delta t_k \\ G_i'(t) = \beta_i G_i(t) \circ e_i - |\beta_i| G_i(t), & 0 \leqslant t \leqslant \Delta t_k, \quad \text{for } i = 1, \ldots, r, \end{array} \right.$$

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

A fully discrete version of the method takes the form

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

At each step of the algorithm, all functions can be updated with the two operations, composition and linear combination, described in Figure 4.2.1. 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 (65) but with less terms in the right hand side. Nevertheless this gain is mitigated by the number of additional fractional steps.

4.2.5 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) $W \geqslant 4$: 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 (65) 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 $W \geqslant 4$.

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 4.11. *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 4.12. *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 4.11 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 4.11, 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 (65). Expansion of e_0 as a linear combination of functions in E_h and repeated application of Proposition 4.12 show that the sum (65) can be reorganized as

$$H_k = \sum_{d=1}^{D} L_d^+ \circ T \circ L_d^- \tag{70}$$

where $L_d^{\pm}$ are affine functions and D is taken large enough, equal to the total number of terms in (65). 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 4.13. *The series (70) 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 (70) 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 $C \geqslant 2\sum_{d=1}^{D} \left\| L_d^+ \circ T \circ L_d^- \right\|_{L^\infty(I)}$ is taken sufficiently large.

Then partial series $h_r = \frac{1}{2} + \frac{1}{C}\sum_{d=1}^{r} L_d^+ \circ T \circ L_d^-$ (for $1 \leqslant r \leqslant D$) 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

$$h_0 = \frac{1}{2}, \quad h_{r+1} = h_r + \frac{1}{C} L_{r+1}^+ \circ T \circ L_{r+1}^- \quad 1 \leqslant r \leqslant D-1. \tag{71}$$

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 8, 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 8 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 (65) which yields $k + 1$ layers with an increasing number of neurons per layers, or with the Network that comes from

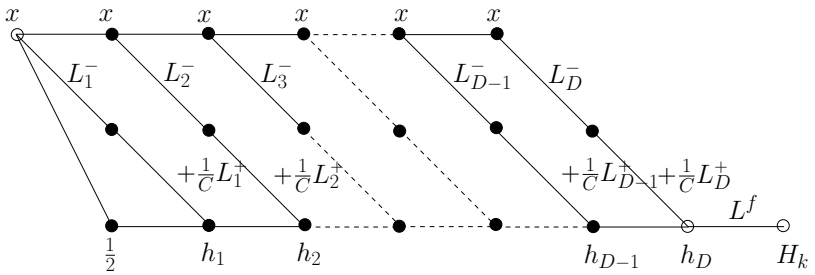

Figure 8: Structure of the Neural Network associated to the formula (70) and to the iterations (71). The top line reproduces x, it is called a source channel. The bottom line is the collection channel which realizes the addition in (71), it is called a collation channel. The intermediate lines correspond 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.

Theorem 4.13 which has a large number of layers and 3 neurons per layer. One can also implement (70) 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.

It can be valuable 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.

4.2.6 Numerical examples

It is possible to calculate analytically all coefficients β_i and γ_i by solving the linear system (61), and to compare numerically the numerical error with the theoretical rate of convergence (66), as done in Table 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 $1 \leqslant i \leqslant r$, with r conveniently chosen. We formulate training as the problem of finding the best coefficients β_i for $1 \leqslant i \leqslant r$ and $e_0 \in V_h$ by minimizing the L^2 cost function

$$W = (e_0, \beta_1, \ldots, \beta_r) \mapsto J(W) = \|H_k(W) - H\|_{L^2(I)}\,, \tag{72}$$

where the integral (in the L^2 norm) is evaluated by quadrature and the basis functions e_i for $i \geqslant 1$ 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[8] in Julia (see `https://julialang.org`) using an automatic

[8]Source code is available at `https://doi.org/10.5281/zenodo.3936433`. It implements the iterations (64) with

differentiation library [16] and an optimization library [14]. The integral in (72) 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 $0 \leqslant x_i = \frac{i-1}{N-1} \leqslant 1$. 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 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 4).

For the tests in Tables 1 and 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 $\alpha \approx \frac{1}{4} < \min_{2 \leqslant k \leqslant 8} K(k)$, 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 1: The polynomial is $H(x) = x - x^2$. One observes $E(k) = O(4^{-k})$ in accordance with the theoretical prediction issued from (55).

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

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 4, where we also provide the accuracy with the Neural Network with the exact coefficients (obtained by solving the linear systems (61)) and the exact constant K. One observes convergence of the training at a better rate than the theoretical prediction.

the two operations described in Figure 4.2.1.

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

Other tests have been made. Two difficulties were observed: the constant $K(k)$ can be greater than one as in Table 4, which makes the results more difficult to interpret; or the time of training with a large number of basis functions ($\geqslant 10$) becomes important with our current implementation. Learning the basis functions $(e_i)_{i\geqslant 1}$ could be investigated to see if some improvements are possible.

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
$E_{\text{ex}}(k)$	8.4e-1	6.3e-1	4.6e-1	3.3e-1
$K_{\text{ex}}(k)$		0.94	0.94	0.94

Table 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: $E(k) \leqslant E_{\text{ex}}(k)$.

5 Datasets

Hereafter, we discuss basic but necessary considerations for the construction of datasets like (3). The two main issues are the curse of dimension, and the answer with the Monte-Carlo method.

A very general framework is to consider (hidden) parameters denoted as $\theta \in \mathbb{R}^p$ and two (explicit) functions $u : \mathbb{R}^p \to \mathbb{R}^m$ and $v : \mathbb{R}^p \to \mathbb{R}^n$. The sampling of the parameters

$$\theta_i \in \mathbb{R}^p, \quad i = 1, 2, \dots$$

defines the interpolation pairs $(x_i, y_i) \in \mathcal{D}$ with

$$x_i = u(\theta_i) \text{ and } y_i = v(\theta_i).$$

The aim is at establishing a causal connection (that is a function) between the x_i's and the y_i's.

Lemma 5.1. *If u admits a left inverse u^{-1}, then one can take*

$$f^{\text{obj}} = vou^{-1}.$$

The left inverse might not exist as well defined function, but this is not the real problem for applications but the curse of dimension. The curse of dimension corresponds principally to $p >> 1$. Usually, n can be moderate because it corresponds to the characterization.

To understand the curse of dimension, it is sufficient to imagine uniform sampling of $\theta \in [0,1]^p$ with q points per dimension. With this strategy to construct the dataset, one gets a number of samples

$$\#(\mathcal{D}) = q^p.$$

Make the hypothesis that one takes 10 points per direction, it yields $\#(\mathcal{D}) = 10^p$. It is sufficient to remember the number of atoms per mole (equal to the Avogadro number 6×10^{23}), the number of bacteria on earth ($\approx 10^{30}$) or the number of particules in the universe ($\approx 10^{80}$, to understand that a brute force discretization is useless if $p >> 1$.

5.1 Distance between points in $\mathbb{R}^m$ with m large

This part is taken from [12].

To be more precise than the above basic observation, consider a function $g \in W^{1,\infty}[0,1]^m$ and simplify by considering $x = \theta$, that is $u = I_{\mathbb{R}^m}$. A finite dataset is considered $\mathcal{D} = \{(x_i, y_i), i = 1, \dots, I\}$. We define $\widetilde{g} : \mathbb{R}^m \to \mathbb{R}^n$ by

$$\widetilde{g}(x) = g(x_i)$$

where i is the index of one of the the closest neighbor of x, that is

$$|x - x_i| \leqslant |x - x_j| \quad j = 1, \dots, I$$

Beware that x_i is not necessarily unique. If there are multiple closest neighbors, one can take the one with the smallest index i. Set

$$\mathrm{Lip}(\alpha) = \left\{ h \in W^{1,\infty}[0,1]^m, \quad \|\nabla h\|_{L^\infty[0,1]^m} \leqslant \alpha \right\}, \quad \alpha > 0.$$

Lemma 5.2. *One has*

$$\sup_{g \in \mathrm{Lip}(\alpha)} \|g - \widetilde{g}\|_{L^\infty[0,1]^p} = \alpha\varepsilon, \qquad \textit{with } \varepsilon = \sup_{x \in [0,1]^m} \min_{1 \leqslant i \leqslant I} |x - x_i|.$$

Proof. By definition

$$|g(x) - \widetilde{g}(x)| = |g(x) - g(x_i)| \leqslant \|\nabla g\|_{L^\infty[0,1]^m} |x - x_i| = \alpha \min_i |x - x_i| \leqslant \alpha\varepsilon.$$

So $\|g - \widetilde{g}\|_{L^\infty[0,1]^m} \leqslant \alpha\varepsilon$.

It remains to show that the bound is reached. The function $\varphi : x \mapsto \min_{1 \leqslant i \leqslant I} \|x - x_i\|$ is continuous (belongs to $C^0[0,1]^m$). Since Ω is compact, then there exists $x^* \in [0,1]^m$ such that $\varphi(x^*) = \max_{x \in [0,1]^m} \varphi(x) = \varepsilon$ (the sup is a max). Denoting a best neighbor x_i^*, one has

$$\min_{1 \leqslant i \leqslant I} |x - x_i| \leqslant |x^* - x_i^*| = \varepsilon \leqslant |x^* - x_i|, \qquad \forall x \in [0,1]^m, \ \forall i = 1, \dots, I.$$

Consider $g^*(x) = \alpha |x - x_i^*|$. It is not difficult to show that $g^* \in \mathrm{Lip}(\alpha)$, which yields of course that $\|g^* - \widetilde{g}^*\|_{L^\infty[0,1]^p} \leqslant \alpha\varepsilon$. Moreover

$$|g^*(x^*) - \widetilde{g}^*(x^*)| = \alpha |x^* - x_i^*| - \alpha |x_i^* - x_i^*| = \alpha |x^* - x_i^*| = \alpha\varepsilon.$$

Therefore

$$\|g^* - \widetilde{g}^*\|_{L^\infty[0,1]^m} \geqslant \alpha\varepsilon.$$

The double inequality yields the claim. □

Note that $\varepsilon = \sup_{x\in[0,1]^m} \min_{1\leqslant i\leqslant I} |x - x_i|$ mans that all $x \in [0,1]^m$ are in balls centered on x_i (a priori $x_i \neq x_i'$ for $x \neq x'$), all balls of radius less or equal to ε. So it is equivalent to say that $[0,1]^m$ is covered by I balls of radius ε, that is $[0,1]^m \subset \cup_i B(x_i, \varepsilon)$. The question is the relation between I and ε, in dimension m. Let us call

$$I^*(\varepsilon) = \min I, \quad \text{where } [0,1]^m \text{ is covered by } I \text{ balls of of radius less of equal to } \varepsilon.$$

The best strategy for the distribution of the x_i is to try to cover $[0,1]^m$ with balls of radius $\varepsilon > 0$, trying at the same time to have the minimal number of such balls.

Lemma 5.3. *One has* $I^*(\varepsilon) \leqslant \left(\frac{\sqrt{m}}{2\varepsilon}\right)^m$.

Proof. By definition, $I^*(\varepsilon) \leqslant I$ where I is a uniform sampling in all directions of hypercubes of size $\Delta x > 0$, each of them **exactly embedded** in balls of radius ε. One has $I\Delta x^m = 1$. Also the corner is at distance $l = \sqrt{m\frac{\Delta x^2}{4}} = \frac{\sqrt{m}}{2}\Delta x$, so one takes $l = \varepsilon$. It yields the proof. □

Lemma 5.4. *One has*

$$\left[\left(\frac{2}{\pi e}\right)^{\frac{m}{2}} \sqrt{\pi m}\left(1 + O(m^{-1})\right)\right]\left(\frac{\sqrt{m}}{2\varepsilon}\right)^m \leqslant I^*(\varepsilon).$$

Proof. Consider an optimal distribution. Since $[0,1]^m \subset \cup_i B(x_i, \varepsilon)$, then

$$1 = \mathrm{Vol}\left([0,1]^m\right) \leqslant I^*(\varepsilon)\mathrm{Vol}\left(B(0,\varepsilon)\right).$$

In even dimension one has $\mathrm{Vol}\left(B(0,\varepsilon)\right) = \frac{\pi^{\frac{m}{2}}}{(m/2)!}\varepsilon^m$. The Stirling formula for large n is $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + O(n^{-1})\right)$. So

$$\frac{1}{\mathrm{Vol}\left(B(0,\varepsilon)\right)} = \left(\frac{\sqrt{m}}{2\varepsilon}\right)^m \left(\frac{2}{\pi e}\right)^{\frac{m}{2}} \sqrt{\pi m}\left(1 + O(m^{-1})\right)$$

which is the low bound of the claim. □

Corollary 5.5. *One has*

$$\left[\sqrt{\frac{2}{\pi e}}\left(1 + O\left(\frac{\log m}{m}\right)\right)\right]\frac{\sqrt{m}}{2} I^*(\varepsilon)^{-\frac{1}{m}} \leqslant \varepsilon \leqslant \frac{\sqrt{m}}{2} I^*(\varepsilon)^{-\frac{1}{m}}.$$

These inequalities explain that the optimal packing is still extremely sensitive to the dimension. For high dimension m, ε cannot be taken as small as desired, because of the cost of sampling (one again, even with optimal sampling).

5.2 Approximating integrals in $\mathbb{R}^m$ with m large

Fortunately, loss functions or error functions like (6) can be interpreted as approximation of integrals for which Monte-Carlo methods are so important. It is advised to consult a dedicated course for more material on this import topic.

The basic ideas can be presented as follows. Consider that a sum can be understood as an approximation of an integral written as

$$\mu = \int_{[0,1]^m} f(x)dx$$

so that

$$J_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

represents an approximation of I. The question is to propose a construction for the x_i's.

Definition 5.6. *The Monte-Carlo writes as follows: choose x_1 at random, then x_2, and so on and so forth.*

In a computer, it is sufficient to use the pre-defined function **random** to realize this idea.

To give a probabilistic meaning of "at random", we make the assumption that there exists another function defined from an abstract space Ω equipped with a measure such that $\int_\Omega d\omega = 1$. Writing $x_1 = X_1(\omega)$ means that the value of x_1 chosen accordingly to the choice of $\omega \in \Omega$. The probability that $x_1 \in A \subset [0,1]^m$ is denoted

$$P\{X_1 \in A\} = \int_{\omega \text{ such that } X_1(\omega)\in A} d\omega.$$

We make the hypothesis that the choice of x_1 does not influence the choice of x_2, x_3, and so on and so forth. More generally, the choice of x_r does not influence the choice of x_s for $s > r$. We note that one can also take $\Omega = [0,1]^m$. Let us continue with this choice which simplifies a lot.

Let us note

$$\sigma = \int_{[0,1]^m} |f(x) - \mu|^2 dx = \int_{[0,1]^m} f(x)^2 dx - \mu^2.$$

Consider that

$$\sigma_N^2 = \int_{([0,1]^m)^N} |J_N - \mu|^2 dx_1 \dots dx_N$$

Lemma 5.7. *One has $\sigma_N = \frac{1}{\sqrt{N}}\sigma$.*

Proof. Direct calculation. □

Lemma 5.8 (Tchebycheff inequality)**.** *Take $\lambda > 0$. Then $P\{|f(X) - \mu_X| \geqslant \lambda\} \leqslant \frac{\sigma}{\lambda}$.*

Theorem 5.9. *One has $P\{|J_N - \mu_X| \leqslant \lambda\} \geqslant 1 - \frac{\sigma}{\sqrt{N}\lambda}$.*

Proof. Use the two previous lemmas. □

5.2.1 An example

Consider the numerical approximation of

$$f^{\text{obj}}(x) = sin(x^1) + e^{-x^2} + x^1(x^2)^2(x^3)^3, \quad x = (x^1, x^2, x^3) \in [0,1],$$

with the least square method, so one takes $f_{W,b}(x_1, x_2, x_3) = Wx+b$. Assume that for some reasons, the least square integral admits a weight

$$J(W,b) = \int_{[0,1]^3} \left| f^{\text{obj}}(x) - f_{W,b}(x) \right|^2 \frac{dx_1}{\sqrt{x^1(1-x^1)}} \frac{dx^2}{\sqrt{x^2(1-x^2)}} \frac{dx^2}{\sqrt{x^3(1-x^3)}}$$

give more importance to the approximation near the boundary of the cube.

To model this with the Monte-Carlo method, consider $\theta = (\theta^1, \theta^2, \theta^3) \in [0,1]^3$ and the transformation $\theta \mapsto x = X(\theta)$

$$x^r = \frac{1}{2}(1 - \cos\theta^r), \quad r = 1, 2, 3.$$

The differential identity $dx^r = \frac{1}{2}\sin\theta^r d\theta^r$ is also

$$jjj$$

Sample uniformly in $[0,1]^3$ $\theta_i \in [0,1]^3$ for $i = 1, \ldots, I$. Construct the dataset $\mathcal{D} = \{(x_i, y_i),\ 1 \leqslant i \leqslant I\}$ with $x_i = X(\theta_i)$ and $y_i = f^{\text{obj}}(x_i)$. Then the Monte-Carlo realization of J is

$$J_I(W,b) = \frac{1}{I} \sum_{i=1}^{I} \left| f(x_i) - y_i \right|^2.$$

By theorem, $J_I(W,b)$ is the MC/probabilistic approximation of

$$K(W,b) = \int_{[0,1]^3} \left| f^{\text{obj}}(X(\theta)) - f(X(\theta)) \right|^2 d\theta.$$

And indeed $K(W,b) = J(W,b)$.

It remains to calculate a pair (W,b) which minimizes $J_I(W,b)$. This is considered hereafter.

6 Stochastic gradient methods and effective ML approximation

We discuss some issues which concern the minimization of the objective function written as

$$W \mapsto J(W) = \sum_{i=1}^{N} j(W, X_i),$$

where j is assembled with either the ReLU or the sigmoid function The parameter/unknown lives in $\Omega \subset \mathbb{R}^p$. We recall the abstract Theorem.

Theorem 6.1. *Let $J : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ be a continuous function over the closed bounded set $\Omega \subset \mathbb{R}^d$ in finite dimension. Then there exists $W_* \in \Omega$ such that $J(W_*) \leqslant J(W)$ for all $W \in \Omega$.*

6.1 Algorithms

All algorithms can be written as variation around the main theme, which is the steepest gradient method.

6.1.1 Steepest gradient method

It writes

$$W^{n+1} = W^n - \Delta t \nabla J(W^n), \sigma > 0. \tag{73}$$

Here with need that J is differentiable and it is better to assume that ∇J is Lipschitz. Indeed the continuous version of (81) introduces a fictitious time variable $t > 0$

$$\begin{cases} W'(t) = -\nabla J(W(t)), & t > 0, \\ W(t) = W_0. \end{cases} \tag{74}$$

Theorem 6.2 (Cauchy-Lipschitz theorem)**.** *Assume ∇J is Lipschitz. Then there exists a maximal time interval $I = [0, T)$ such that the Cauchy problem has a unique solution W in the interval I.*

One has $\frac{d}{dt} J(W(t)) \leqslant 0$, so the value of the functional decreases. Let us assume that J is a convex functional such that

$$\beta \geqslant \nabla^2 J \geqslant \alpha > 0$$

and that W_* is an extremal point

$$\nabla J(W_*) = 0.$$

Then W_* is a minimizer since one has for all W

$$J(W) = J(W_*) + \nabla J(W_*) \cdot (W - W_*) + \frac{1}{2} \nabla J(W_* + \mu(W - W_*))(W - W_*) \cdot (W - W_*) \geqslant J(W_*).$$

Proposition 6.3. *One has*

$$|W(t) - W_*| \leqslant e^{-\alpha t/2}.$$

Proof. Indeed

$$\frac{d}{dt} |W(t) - W_*|^2 = 2W'(t) \cdot (W - W_*) = -2 (\nabla J(W(t))) \cdot (W - W_*)$$

$$= -2 (\nabla J(W(t)) - \nabla J(W_*)) \cdot (W - W_*) \leqslant -\alpha |W(t) - W_*|^2 .$$

□

Now we consider the discrete algorithm (81).

Proposition 6.4. *Assume $\alpha < \frac{2}{\beta}$ and $W^n \neq W_*$. Then $J(W^{n+1}) < J(W^n)$. Moreover $\lim_{n\to\infty} W^n = W_*$.*

Proof. Indeed

$$J(W^{n+1}) = J(W^n) + \nabla J(W^n) \cdot (W^{n+1} - W^n) + \frac{1}{2} \nabla J(W_* + \mu(W - W_*))(W - W_*) \cdot (W - W_*)$$

$$= J(W^n) - \alpha |\nabla J(W^n)|^2 + \frac{1}{2} \alpha^2 \nabla J(W_* + \mu(W - W_*)) \nabla J(W^n) \cdot \nabla J(W^n)$$

$$\leqslant J(W^n) - \left(\alpha - \frac{1}{2}\alpha^2\beta\right)|\nabla J(W^n)|^2 .$$

Take $0 < \alpha = \frac{2}{\beta} - \varepsilon$ so $\sigma = \alpha - \frac{1}{2}\alpha^2\beta > 0$. Then $J(W^{n+1}) \leqslant J(W^n) - \sigma|\nabla J(W^n)|^2$. So either $W^n \neq W_*$, then $\nabla J(W^n) \neq 0$ and $J(W^{n+1}) < J(W^n)$: or $W^n = W_*$ and the convergence is reached. □

6.1.2 A weak steepest gradient method (for the ReLU activation function)

The ReLU function is such that ∇J is bounded but discontinuous, so it cannot be Lipschitz. So one can ask what is the meaning of the method. We show that a convenient adaptation of weak method is possible, where all derivatives are eliminated.

One starts by considering that J is continuous, with a first derivative bounded in L^∞ and convex. We also assume that J can be approximated by a smooth function J_ε, which is continuous, convex and C^2. For this, it is sufficient to regularize the ReLU function (like for a Kruzkov entropy). Consider

$$W'_\varepsilon(t) = -\nabla J_\varepsilon(W_\varepsilon(t)). \tag{75}$$

Lemma 6.5. *Assume W is a C^1 function. One has*

$$W'_\varepsilon(t)\cdot(Y - W_\varepsilon(t)) + J_\varepsilon(Y) - J_\varepsilon(W_\varepsilon(t)) \geqslant 0$$

for all Y and all t.

Proof. Indeed

$$\begin{aligned} &W'_\varepsilon(t)\cdot(Y - W_\varepsilon(t)) + J_\varepsilon(Y) - J_\varepsilon(W_\varepsilon(t)) \\ &= -\nabla J_\varepsilon(W_\varepsilon(t))\cdot(Y - W_\varepsilon(t)) + J_\varepsilon(Y) - J_\varepsilon(W_\varepsilon(t)) \geqslant 0. \end{aligned}$$

□

Passing to the limit one gets

$$W'(t)\cdot(Y - W(t)) + J(Y) - J(W(t)) \geqslant 0 \tag{76}$$

for all Y and all t. In this formulation the derivative ∇J is eliminated. It is possible to take

$$W \in W^{1,\infty}[0,T)$$

because the boundedness of ∇J_ε yields the uniform bound

$$\|W_\varepsilon\|_{W^{1,\infty}[0,T)} \leqslant C. \tag{77}$$

It is possible to eliminate also the derivative in time. Take $\varphi \in C_0^{1,+}(0,T)$ a non negative function with compact support and a continuous derivative. Then a weak formulation is

$$\int_0^T \left[\frac{|W_\varepsilon(t) - Y|^2}{2}\varphi'(t) + (J_\varepsilon(Y) - J_\varepsilon(W_\varepsilon(t)))\,\varphi(t)\right] dt \geqslant 0$$

for all $\varphi \in C_0^{1,+}(0,T)$. Passing to the limit one gets

$$\int_0^T \left[\frac{|W(t)-Y|^2}{2} \varphi'(t) + (J(Y) - J(W(t)))\, \varphi(t) \right] dt \geqslant 0 \tag{78}$$

for all $\varphi \in C_0^{1,+}(0,T)$ and all Y. The initial data is $W(0) = W_0$. Now the unknown function W needs only minimal smoothness, for example $W \in C^0[0,T)$.

Lemma 6.6. *There exists one (or more) solution to (78).*

Proof. By compactness from (75) and (77). □

Lemma 6.7. *One has $W(0) = W_0$.*

Proof. Take $\varphi(t) = (1 - t/\varepsilon)_+$ and pass to the limit. □

Lemma 6.8. *The solution of $W \in C^0[0,T)$ of (78) with initial data $W(0) = W_0$ is unique.*

Proof. The proof is an adaptation of the method of doubling of unknowns of Kruzkov. □

Take the particular example $w \in \mathbb{R}$ and

$$J(w) = |w|.$$

Lemma 6.9. *The solution of (78) is defined for $t \in \mathbb{R}^+$. It is*

$$w(t) = w_0(1-t)_+.$$

Proof. Check. □

A possible implicit discretization of (76) is

$$\left(W^{n+1} - W^n\right) \cdot \left(Y - W^{n+1}\right) + \sigma \left(J(Y) - J(W^{n+1})\right)\big) \geqslant 0, \quad \forall Y. \tag{79}$$

Lemma 6.10. *W^{n+1} is unique.*

Proof. We show the uniqueness. Assume two solutions $i = 1, 2$

$$\left(W_i^{n+1} - W^n\right) \cdot \left(Y - W_i^{n+1}\right) + \sigma \left(J(Y) - J(W_i^{n+1})\right)\big) \geqslant 0, \quad \forall Y.$$

Then

$$\left(W_1^{n+1} - W^n\right) \cdot \left(W_2^{n+1} - W_1^{n+1}\right) + \sigma \left(J(W_2^{n+1}) - J(W_1^{n+1})\right)\big) \geqslant 0$$

and

$$\left(W_2^{n+1} - W^n\right) \cdot \left(W_1^{n+1} - W_2^{n+1}\right) + \sigma \left(J(W_1^{n+1}) - J(W_2^{n+1})\right)\big) \geqslant 0.$$

By summation

$$-\left|W_1^{n+1} - W_2^{n+1}\right|^2 \geqslant 0.$$

□

Lemma 6.11. *W^{n+1} exists.*

Proof. Take J_ε a C^2 convex regularization of (79) and consider the inequalities

$$(Z_\varepsilon - W^n) \cdot (Y - Z_\varepsilon) + \sigma \left(J(Y) - J(Z_\varepsilon) \right) \geqslant 0, \quad \forall Y.$$

Since J_ε is convexe, it is equivalent to

$$Z_\varepsilon - W^n + \sigma \nabla J_\varepsilon(Z_\varepsilon) = 0.$$

The vector Z_ε is the minimum of the convex problem

$$Z_\varepsilon = \text{argmin}_Q \left(\frac{1}{2} |Q - W^n|^2 + \sigma J_\varepsilon(Q) \right).$$

Then pass to the limit $\varepsilon = 0$. □

Lemma 6.12. $J(W^{n+1}) \leqslant J(W^n)$

Proof. Take $Y = W^n$. □

6.2 Stochastic gradients

Now we consider algorithms which render the ML approximation of functions effective.

A stochastic gradient algorithm introduce some degree of stochasticity with at least two objectives: firstly to escape local minima in which a classical descent algorithm can be trapped; secondly to visit the database in a way which allows to deal with large ones. These two different objectives are described separately.

6.2.1 Momentum methods

One add at least one additional variable, let us call it Z, and consider the continuous in time method

$$t > 0 : \qquad \begin{cases} W'(t) = Z(t), \\ Z'(t) = -\nabla J(W(t)). \end{cases} \tag{80}$$

A possible discrete version writes

$$\begin{cases} W^{n+1} = W^n + \sigma Z^n, \sigma > 0, \\ Z^{n+1} = Z^n - \sigma \nabla J(W^n) \end{cases} \tag{81}$$

We will assume that J is a C^2 function, so that ∇J is C^1.

An elementary property follows.

Lemma 6.13. *One has*

$$\frac{d}{dt} \left(J(W(t)) + \frac{1}{2} |Z(t)|^2 \right) = 0.$$

Starting with $W(0) = W_0$ and $Z(0) = 0$, this method is able to visit a database for some W such that $J(W) \leqslant J(W_0)$. If $J(W) < J(W_0)$, then $Z \neq 0$. Therefore the advantage of this approach is that it is not trapped in local minima, while the disadvantage is that it is non stationary even at a global minimum.

Various methods have been proposed to introduce some "friction". The Nesterov method can be summarized as

$$t > 0 : \qquad \begin{cases} W'(t) = Z(t), \\ Z'(t) = -\nabla J(W(t) + \nu Z(t)). \end{cases} \tag{82}$$

Lemma 6.14. *One has*

$$\frac{d}{dt}\left(J(W(t)+\nu Z(t))+\frac{1}{2}|Z(t)|^2\right)=-\nu\,|\nabla J(W(t)+\nu Z(t))|^2\leqslant 0$$

for $\nu\geqslant 0$.

So $\nu Z(t)$ can be interpreted as certain kind of friction.

Another method is as follows, where $\nu\geqslant 0$ and $\varepsilon\geqslant 0$.

$$t>0:\qquad \left\{\begin{array}{l} W'(t)=Z(t),\\ Z'(t)=-\nabla J(W(t))-\frac{\nu}{\sqrt{|Z'(t)|^2+\varepsilon}}Z'(t).\end{array}\right. \tag{83}$$

Lemma 6.15. *One has*

$$\frac{d}{dt}\left(J(W(t))+\frac{1}{2}|Z(t)|^2\right)=-\nu\frac{|Z'(t)|^2}{\sqrt{|Z'(t)|^2+\varepsilon}}\leqslant 0.$$

So

$$\int_0^\infty\frac{|Z'(t)|^2}{\sqrt{|Z'(t)|^2+\varepsilon}}dt<\infty.$$

Assuming that J is bounded from below, then $|Z(t)|$ is bounded from above

$$|Z(t)|\leqslant C,\quad t\geqslant 0.$$

6.2.2 Nesterov acceleration

6.2.3 Batches

Assume

$$J(W)=\sum_{i=1}^N|\varphi(W:X_i)-y_i|^2$$

and $N>>1$ is large. Then all steps of the gradient methods evoked above are very costly, just because the evaluation of the gradient ∇J is very costly at all steps.

A batch decomposition of the stack $\{X_i\}_{1\leqslant i\leqslant N}$ writes as

$$\{X_i\}_{1\leqslant i\leqslant N}=\cup_{r=1}^p\mathcal{B}_r$$

where $\mathcal{B}_r=\{X_i\}_{(r-1)q+1\leqslant i\leqslant rq}$ is the r-th batch and $pq=N$. These batches are decided at random, that is stochastically. One can write

$$J(W)=\sum_{r=1}^p J_r(W).$$

Decide of a small time step $\Delta t>0$. One can consider the splitting algorithm where $n=0,1,\ldots$ and

$$\begin{array}{l} Z_0(\Delta t)=W^n\\ r=0,\ldots,p:\left\{\begin{array}{l} Z_r(0)=Z_{r-1}(\Delta t)\\ Z_r'(t)=-\nabla J_r(Z_r(t)),\quad 0<t\leqslant\Delta t,\end{array}\right.\\ W^{n+1}=Z_p(\Delta t).\end{array}$$

A discrete version writes

$$\begin{array}{l} Z_0(\Delta t) = W^n \\ r = 0, \dots, p: \;\; Z_r(\Delta t) = Z_r(0) - \Delta t^n \nabla J_r(Z_r(0)), \\ W^{n+1} = Z_p(\Delta t). \end{array}$$

We make an hypothesis which is locally relevant around any local strict minimum of J.

Theorem 6.16. *Assume J is quadratic strictly convex, with minimum W_*. Assume J made with batches which are also quadratic convex. Assume*

$$\lim_{n\to\infty} \Delta t^n = 0 \text{ and } \sum_n \Delta t^n = \infty.$$

Then

$$\lim_{n\to\infty} \Delta W^n = W_*,$$

and this is independent of the batches.

Proof. To write. □

Theorem 6.17. *The same one for ADAM optimizer with friction.*

Proof. To write. □

Theorem 6.18. *The same one for Nesterov acceleration.*

Proof. To write. □

7 Applications

7.1 Softwares

The success of modern machine learning methods and deep learning methods is mainly due to the quality of the softwares available. There are many: Tensorflow (Google), Scikit-Learn (Inria), Pytorch (Facebook), Julia (MIT), Matlab (Mathworks), ... They are very flexible and have more or less the same features, even if the presentation might differ. The coding follow the same kind of natural language like Python (Tensorflow).

Matlab is free on access on the Sorbonne University site. Documentation is at `https://fr.mathworks.com/help/matlab/`.

`code Matlab`

Julia is free, developed by MIT.

`code Julia`

`toto`

Here is an example which calculates a dataset for the Takagi function.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
from random import seed
import shutil
```

```
import math
import string
import numpy
import numpy as np
seed()

print("-------------")
print ("Fonction de Takagi= ")
print("-------------")

M=13000

string_file='%5.8f,%5.8f,%5.4f'
string_file=string_file+' \n'
strstr_train='./training-images/0/TRAIN_tak.txt'
fichier_train = open(strstr_train,'w')
strstr_test='./training-images/0/TEST_tak.txt'
fichier_test  = open(strstr_test,'w')

for i in range(0,M):
    a=i/(M-1.)              #   a=np.random.uniform(0,1)
    b= a**2                 #(a-a*a-1./6)*6. # np.cos(2*3.1415926*8*a)
    eps=0.00
    if (np.random.uniform(0., 1.)>0.2):
            fichier_train.write(string_file %  (a,b+np.random.uniform(-eps,eps),a) )
    else:
            fichier_test.write(string_file %  (a,b,a) )

fichier_train.close()
fichier_test.close()
print("This is the end")
```

7.2 Calculating gradients and back propagation

To be able to calculate gradients of composed functions, the most important formula is the chain rule.

Notation 7.1. *In this section (and only in this section), we adopt the following conventions.*
• *The gradient of* $u \in C^1\left(\mathbb{R}^a : \mathbb{R}^b\right)$ *is a matrix with* b *lines and* a *columns*

$$\nabla u = \begin{pmatrix} \partial_1 u_1 & \partial_2 u_1 & \dots & \partial_a u_1 \\ \partial_1 u_2 & \partial_2 u_2 & \dots & \partial_a u_2 \\ \dots & \dots & \dots & \dots \\ \partial_1 u_b & \partial_2 u_b & \dots & \partial_a u_b \end{pmatrix} \in \mathcal{M}_{b,a}(\mathbb{R}).$$

- *The gradient of* $\psi \in C^1\left(\mathcal{M}_{a,b}(\mathbb{R}) : \mathbb{R}\right)$ *is also a matrix with* b *lines and* a *columns*

$$\nabla\psi = \begin{pmatrix} \partial_{(11)}\psi & \partial_{(21)}\psi & \dots & \partial_{(a1)}\psi \\ \partial_{(12)}\psi & \partial_{(22)}\psi & \dots & \partial_{(a2)}\psi \\ \dots & \dots & \dots & \dots \\ \partial_{(1b)}\psi & \partial_{(2b)}\psi & \dots & \partial_{(ab)}\psi \end{pmatrix} \in \mathcal{M}_{b,a}(\mathbb{R}) = \mathcal{M}_{a,b}(\mathbb{R})^t.$$

Remark 7.2. *Take* $b = 1$. *Then both notations coincide. The gradient of a scalar function is a flat vector in* $\mathcal{M}_{1a}(\mathbb{R}) = (\mathbb{R}^a)^t$.

Let $u \in C^1\left(\mathbb{R}^a : \mathbb{R}^b\right)$ and $v \in C^1\left(\mathbb{R}^b : \mathbb{R}^c\right)$. Then the gradient of $w = vou$ is given by the chain rule which is the standard multiplication $\times$ of matrices thanks to our notations

$$\nabla w = \underbrace{\nabla v}_{\in \mathcal{M}_{c,b}(\mathbb{R})} \times \underbrace{\nabla u}_{\in \mathcal{M}_{b,a}(\mathbb{R})} \in \mathcal{M}_{c,a}(\mathbb{R}). \tag{84}$$

Exercise 7.3. *Prove (84).*

Take $f \in C^1\left(\mathbb{R}^m : \mathbb{R}^n\right)$ given by (14) with σ a sigmoid which is noted

$$(x \mid W^{p+1}, \dots W^0 : b^{p+1}, \dots, b^0) \longmapsto f(x \mid W^{p+1}, \dots W^0 : b^{p+1}, \dots, b^0).$$

Take a real valued function $\varphi \in C^1\left(\mathbb{R}^n : \mathbb{R}\right)$. We use the notation

$$\nabla\varphi = (\nabla\varphi of)(x \mid W^{p+1}, \dots W^0 : b^{p+1}, \dots, b^0) \in \mathcal{M}_{1,n}(\mathbb{R}).$$

Lemma 7.4. *The derivative of* φof *with respect to* x *is*

$$\nabla_x \varphi of = \nabla\varphi \times W^{p+1} \times \nabla\sigma \times W^p \times \dots \times W^1 \times \nabla\sigma \times W^0 \in (\mathbb{R}^m)^t, \tag{85}$$

where $\nabla\sigma \in \mathcal{M}_{a_r}(\mathbb{R})$ *is a diagonal matrix of size* a_r *for* σ *just before* W^{r+1}.

Lemma 7.5. *The derivative of* φof *with respect to* $b^r \in \mathbb{R}^{a_{r+1}}$ *is*

$$\nabla_x \varphi of = \nabla\varphi \times W^{p+1} \times \nabla\sigma \times W^p \times \dots \times W^{r+1} \times \nabla\sigma \in (\mathbb{R}^{a_{r+1}})^t \tag{86}$$

with the same notation for $\nabla\sigma$.

Lemma 7.6. *The derivative of* φof *with respect to* $W^r \in \mathcal{M}_{a_{r+1},a_r}(\mathbb{R})$ *is*

$$\nabla_{W^r} \varphi of = \underbrace{\sigma(\underbrace{W^{r-1}\sigma(\dots)}_{\in \mathcal{M}_{a_r,1}(\mathbb{R})} + b^{r-1})}_{} \times \underbrace{\left(\underbrace{\nabla\varphi \times W^{p+1} \times \nabla\varphi \cdots \times W^{r+1}}_{\in \mathcal{M}_{1,a_{r+1}}(\mathbb{R})} \times \nabla\sigma\right)}_{} \in \mathcal{M}_{a_r,a_{r+1}}(\mathbb{R}) \tag{87}$$

with the same notation for $\nabla\sigma$.

These notations allow to describe the essence of the **back propagation** method which is used to calculate all the gradients with respect to (W^r, b^r), $1 \leqslant r \leqslant p+1$, needed for the various descent methods.

Definition 7.7. *The back propagation method is the iteration from* $r = p+1$ *to* $r = 0$ *(that is for descending values) which allows to compute the series of matrix-matrix multiplications (86-87).*

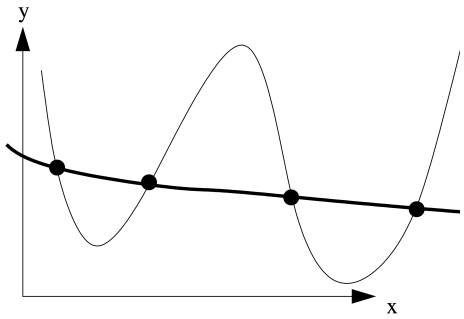

Figure 9: Two curves which pass through interpolation points. The one with over-fitting has stronger gradients.

7.3 Over-fitting

Over-fitting is a common problem. It is related to the fact that a given approximation can be good or excellent at certain points, but bad at neighboring points.

A possibility to tackle this issue is by computing bounds for the derivative of f with respect to x which is a simplification of (85)

$$\nabla_x f = W^{p+1} \times \nabla\sigma \times W^p \times \cdots \times W^1 \times \nabla\sigma \times W^0 \in \mathcal{M}_{n,m}(\mathbb{R}), \tag{88}$$

where $\nabla\sigma \in \mathcal{M}_{a_r}(\mathbb{R})$ is a diagonal matrix of size a_r for σ just before W^{r+1}.

Lemma 7.8. *One has* $\|\nabla_x f\|_{l^\infty} \leqslant \|\sigma'\|^p_{L^\infty(\mathbb{R})} \Pi_{r=0}^{p+1} \|W_r\|_{l^\infty}$ *where* $\|W\|_{l^\infty} = \sup_j \sum_i |w_{ij}|$.

Proof. Standard with such a standard norm for matrices. The matrix $\nabla\sigma$ is diagonal (of various size), so $\|\nabla\sigma\|_{l^\infty} \leqslant \|\sigma'\|_{L^\infty(\mathbb{R})}$. □

Lemma 7.9. *One has* $\|\nabla_x f\|_{l^1} \leqslant \|\sigma'\|^p_{L^\infty(\mathbb{R})} \Pi_{r=0}^{p+1} \|W_r\|_{l^1}$ *where* $\|W\|_{l^1} = \sup_j \sum_i |w_{ij}|$.

Proof. Since $\nabla\sigma$ is diagonal then $\|\nabla\sigma\|_{l^1} = \|\nabla\sigma\|_{l^\infty} \leqslant \|\sigma'\|_{L^\infty(\mathbb{R})}$. □

It yields mechanisms to control over-fitting caused by oscillations by means of proper normalization. These mechanisms are used in software.

Remark 7.10. *A first mechanism to control over-fitting is to choose a sigmoid with bounded derivative* $\|\sigma'\|_{L^\infty(\mathbb{R})} = 1$*, like in (37). One can use the ReLU function as well since* $\|R'\|_{L^\infty(\mathbb{R})} = 1$.

Remark 7.11. *A second mechanism to control over-fitting is to normalize the norm of the matrices* W^r*. This is closely related to what is called* **weights normalization** *in softwares. Typically one decides that* $\|W^r\| \leqslant c$ *for some norm and some constant* $c > 0$*. A priori the weight of output layer* W^{p+1} *is not normalized.*
For a large number $p >> 1$ *of hidden layers, weights normalization is a fundamental procedure to control the sensitivity of* f *with respect to variations of* x*.*

7.4 Applications

7.4.1 Interpolation of functions

and calculation of the discrete derivatives with 4 more non trainable layers of neurons.

Simple sign constraints with non trainable weights (exemple relu with max-value =1 if $f(x) \leqslant 1$).

7.4.2 Classification of numerical oscillations in fluid solver (à la Ray-Hesthaven)

7.4.3 Finite Volume numerical fluxes for interface reconstruction in fluid

7.4.4 MNIST, ImageNet, classification of images

7.4.5 Inverse problems (A la Pironneau, with FreeFem++)

7.4.6 Identification of PDE coefficients (Karniadakis ?)

projects based on papers

7.4.7 POD, based on Karen Wilcox and Maday

7.4.8 Auto-encoder

Detection of the dimension of the curve $(x, y) = f(t) \in \mathbb{R}^2$.

References

[1] G. Allaire, Analyse numérique et optimisation, Éditions de l'École Polytechnique 2005.

[2] Braun, Jürgen ; Griebel, Michael . On a constructive proof of Kolmogorov's superposition theorem. Constr. Approx. 30 (2009), no. 3, 653–675.

[3] H. Brezis,

[4] F. Cholet, Deep-Learning with Python, Manning, 2018.

[5] G. Cybenko, Approximation by Superpositions of a Sigmoidal Function, Math. Control Signals Systems (1989) 2:303-314.

[6] Nonlinear Approximation and (Deep) ReLU Networks I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, arxiv 2019.

[7] B. Després and H. Jourdren, Machine Learning design of Volume of Fluid schemes for compressible flows, JCP 2020.

[8] I. Goodfelllow, Y. Benjio, A. Courville, Deep Learning, 2015.

[9] Hornick,

[10] D. P. Kingma and J. L. Ba, ADAM: a method for stochastic optimization

[11] Kurkova

[12] S. Mallat, online.

[13] W. Rudin,

[14] Sprecher

[15] T. Takagi, A simple example of the continuous function without derivative, Proc. Phys.-Math. Soc. Jpn., vol. 1,? 1901, p. 176?177.

[16] Yarostky,

References

[1] A. Bensoussan, Y. Li, D. P. C. Nguyen, M.-B. Tran, S. C. P. Yam, and X. Zhou. Machine learning and control theory. arXiv:2006.05604.

[2] Mikael Bodèn. A guide to recurrent neural networks and backpropagation. In *the Dallas project, SICS technical report T2002:03*. SICS, 2002.

[3] Philippe 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(4):303–314, 1989.

[5] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and 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 and H Jourdren. Machine learning design of volume of fluid schemes for compressible flows. *Journal of Computational Physics*, 408, 2020.

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

[9] M. Hata and M. Yamaguti. Weierstrass's function and chaos. *Hokkaido Mathematical Joumal*, 12:333–342, 1983.

[10] M. Hata and M. Yamaguti. The takagi function and its generalization. *Japan J. AppL Math.*, 1:83–199, 1984.

[11] Juncai He, L. Li, J. Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *Journal of Computational Mathematics*, 38(3):502–527, 2020.

[12] Bo Li, Shanshan Tang, and Haijun Yu. Better approximations of high dimensional smooth functions by deep neural networks with rectified power units. *Commun. Comput. Phys.*, 27(2):379–411, 2020.

[13] J. Lu, Z. Shen, H. Yang, and S. Zhang. Deep network approximation for smooth functions. `https://blog.nus.edu.sg/matzuows/publications/`, 2020.

[14] Patrick K. Mogensen and Asbjørn N. Riseth. Optim: A mathematical optimization package for julia. *Journal of Open Source Software*, 3(24):615, 2018.

[15] Joost A. A. Opschoor, Philipp C. Petersen, and Christoph Schwab. Deep ReLU networks and high-order finite element methods. *Anal. Appl. (Singap.)*, 18(5):715–770, 2020.

[16] J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in Julia. *arXiv:1607.07892*, 2016.

[17] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 97:103–114, 2017.