Universitat Autònoma de Barcelona

Degree Thesis

Improvements of Deterministic Processes through Neural Networks

---

Author:
Gerard Lahuerta Martín

Supervisor:
Dr. Lluís Alsedà Soler

A thesis submitted in fulfilment of the requirements for the degree of Computational Mathematics and Data Analytics in the

SCIENCE FACULTY

June 2024

Declaration of Authorship

Abstract

Acknowledgements

I want to thank my family for their support throughout my thesis, degree, and life.

I am especially grateful to my mother, Maria Montserrat, for her encouragement and guidance in facing the challenges of this thesis and my career.

Thanks to everyone who treated me as if I was part of their family.

I also appreciate the encouragement and support of my colleagues.

Finally, I want to give a special mention to Dr. Lluís Alsedà for guiding me through the thesis and being an inspiration.

Thank you all for everything.

Contents

List of Figures

List of Tables

Preface

This report is an ambitious project that has been developing for months and has been meditated even more.
In all the process each piece and each result obtained remarks the viability of this type of methodologies to improve the task of neural network development and how it can be implemented to solve problems that have high computational costs.

Introducción

1 Neural Networks

A neural Network is made of individual and independent elements connected between them, passing and managing the information through the network formed.
In this thesis we will focus on one of the simplest networks, a multilayer perceptron, to test the different methods of optimization.

1.1 Multilayer Perceptron and Perceptron neuron

One of the simple Neural Networks to analyse is the Multilayer Perceptron[1].
It was first proposed by Frank Rosenblat[2] in 1958 (nevertheless its approach did not learn either produce accurate results).

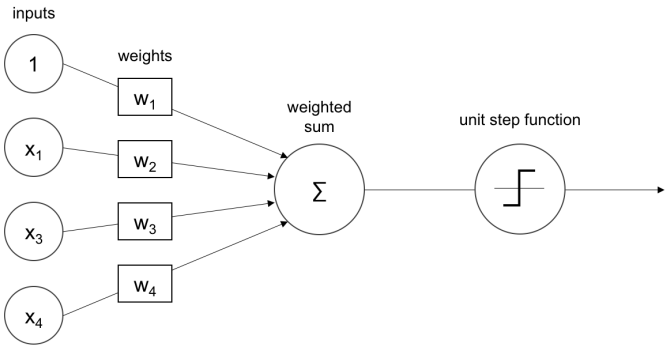


Figure 1: Schema of the Perceptron neuron

The Neural Network is made by individual neurons called *Perceptrons*. The neurons are split into input, weight and activation functions.
Nevertheless, the most important part of the neuron, and that determines significantly the capabilities of the neuron, is the activation function (which returns the output of the neuron).

The traditional activation function used in the Multilayer Perceptron is the Sigmoid:

$$f(x) = \frac{1}{1 + e^{-w \cdot x}}, \text{ where: } x, w \in \mathbb{R}^n$$

The Multilayer Perceptron Topology can be split into layers of three types:

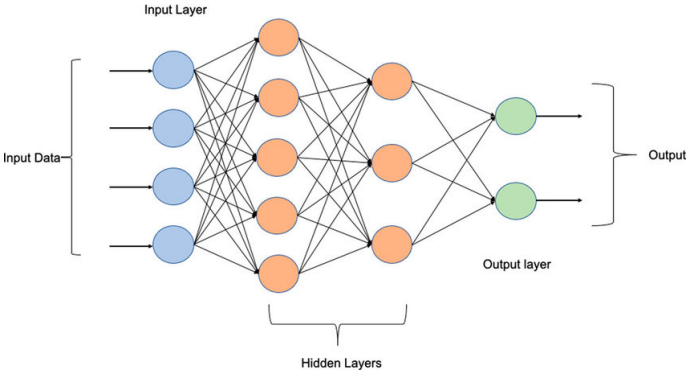


Figure 2: Schema of the Multilayer perceptron

- **Input Layer**: The initial set of neurons of the Multilayer Perceptron.
- **Output Layer**: The final set of neurons of the Multilayer Perceptron.
- **Hidden Layers**: The set of neurons (in layers) between the input and output layers.

[1] To obtain more information about Multilayer Perceptron functionality check the following link
[2] Frank Rosenblat, psychologist and father of deep learning, check its biografy.

1.2 Why Neural Networks works

It is important to remark that what gives us the capacity to model any system is that we can express every problem as a function (no matter if it is a classification task, probability function, prediction, or regression function).
The association between a task and a function allow us to apply the **Universal Approximation Theorem**.

- **Definition:** Universal Approximation Theorem.

For any function $f : \mathbb{R}^n \to \mathbb{R}^m$, with $m, n \in \mathbb{N}$, and a subset $D \subset \mathbb{R}^n$ where f is continuous at all D, $\exists\{(w_i, b_i, c_i)\}_{i=0}^{k}$ that:

$$f(\vec{x}) - \lim_{k\to\infty} \sum_{i=0}^{k} c_i \sigma\left(w_i^T \cdot \vec{x} + b_i\right) = 0$$

Where $w_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$, $c_i \in \mathbb{R}^m$, $\vec{x} \in D$ and σ the sigmoid function.

The parameters $\{w_i, b_i, c_i\}$ are associated (respectively) with the weights, bias and scale factor of the i-th neuron.
Neural Networks essentially use this theorem with a limited number of sigmoids (equal to de number of neurons), consecuently an error of approximation is given.

$$f(\vec{x}) - \sum_{i=0}^{k} c_i \sigma\left(w_i^T \cdot \vec{x} + b_i\right) = \epsilon, k \in \mathbb{N}$$

Modeling systems the information of the function shape/tendency is limited or none, making it only possible to study the problem using data values.
This cases hare handle using the *square-norm* metric (or *Mean Squared Error*).

- **Definition:** Square-norm:

Being $f : \mathbb{R}^n \to \mathbb{R}^m$ function followed by the system. Being $g : \mathbb{R}^n \to \mathbb{R}^m$, with $m, n \in \mathbb{N}$, as $g(\vec{x}) = \sum_{i=0}^{k} c_i \sigma\left(w_i^T \cdot \vec{x} + b_i\right)$.
Given a dataset $B = \{(\vec{x_i}, \vec{y_i})\}_{i=0}^{N}$, with $N \in \mathbb{N}$, where $\forall(\vec{x_i}, \vec{y_i}) \in B, \vec{y_i} = f(\vec{x_i})$.

$$\Delta^2 = \frac{1}{N} \sum_{i=0}^{N} (\vec{y_i} - g(\vec{x_i}))^2$$

Decreassing the error Δ implies reducing the error ϵ because as Δ decrease the model improves its approximation whereas ϵ decrease.
As more neurons are added to the Network, the approximation and the training time increase. This increase in training time for the model in cases is not worth it as it represents a minimum improvement of the model.
Moreover, the Backpropagation[3] method used to train Neural Networks have a complexity that increases the processing time exponentially to the number of neurons.
Therefore, trying to obtain the ideal topology for solving a problem is a difficult task that in many cases resides in trial and error.

[3]The Backpropagation method is an algorithm to modify the internal parameters of a Neural network by using the chain rule, more information in the following link

2 Neural Networks topology

The main objective of the study is to obtain a more efficient process using neural networks than the traditional ways that have been used at the time.
Doing this requires, as has been explained before, a non-understood process to obtain an efficient neural network.
Therefore, an observation has been done while approaching this problem that could partially solve that non-understand of what is doing the neural network and deduce the optimal architecture.

2.1 Hypotesis and Proposal

As has been said before, when approaching this problem an intuitive idea appeared.

- **Proposal:** If there is enough information about the system to model, there exists an efficient neural network in which architecture can be obtained using rules.

The idea proposed is influenced by the following observations:

1. Every deterministic process is a combination of restrictions that can be represented as a function.
2. Every function can be split into domains its behaviour can be categorized as periodical, irrational, or polynomial.
3. The periodical, polynomial and irrational functions are continuous in "all"[4] domain.
4. All continuous functions can be approximated using the **Universal Approximation Theorem**.

Moreover, if a function does not follow one of these types of functions, it can be split by more intervals until each of them can be approximated by one of these types or even can be approximated using *Fourier Series* or *Taylor series* allowing it to be modelled by periodical and polynomial functions.
Therefore, this proposal can be verified by reducing it to corroborate the following points:

1. Each category of functions mentioned can be modelled using a particular architecture of the neural network.
2. Exists a determined minimum number of data registers where the neural network can be trained.

To verify the proposal topological, efficiency and precision studies are required.

[4] In case of irrational functions such as $\sqrt{x}$, it can be interpreted as $f(x) = \begin{cases} 0, x < 0 \\ \sqrt{(x)}, x \geq 0 \end{cases}$

2.2 Study of deterministic process

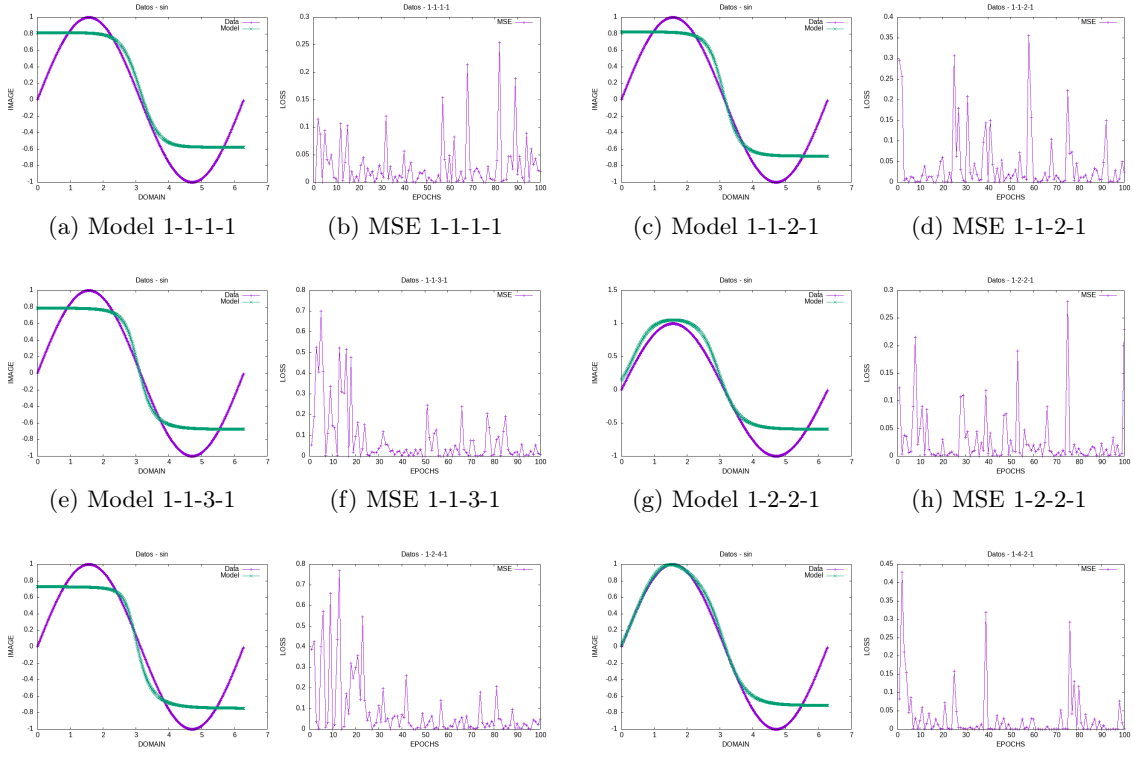
To start, let's assume the following axioms to simplify the study and give some feedback to make the correlations to verify or dismiss the proposal.

1. All periodic functions can be approximated as a combination of sinus functions.
2. All polynomial functions can be approximated as a combination of linear functions.
3. All irrational functions can be approximated as a combination of irrational functions.

All points can be proven using *Taylor Series* and *Fourier Series*. Therefore, the study will be reduced to taking patterns of the architectures used to approximate the functions $\sin(x), \sqrt{(}x), x^2$.
This paper uses $I - H - N - O$ nomenclature to represent the architecture of the neural network, where the letters I, H, N, O represent the number of inputs, the hidden layers, the number of neurons for each hidden layer and the number of outputs respectively.

2.2.1 Study of the *sin* function

To test if there exists an architecture that can approximate efficiently the sin function in the real plane, will be studied to approximate the function in the interval $[0, 2\pi]$ using a fixed number of iterations with different architectures.



(a) Model 1-1-1-1 (b) MSE 1-1-1-1 (c) Model 1-1-2-1 (d) MSE 1-1-2-1

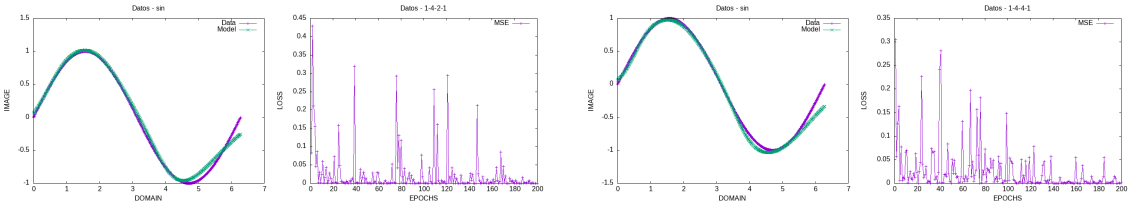
(e) Model 1-1-3-1 (f) MSE 1-1-3-1 (g) Model 1-2-2-1 (h) MSE 1-2-2-1

(i) Model 1-2-4-1 (j) MSE 1-2-4-1 (k) Model 1-4-2-1 (l) MSE 1-4-2-1

Figure 3: Results of different architectures approximating the function *sin*

The results shown in Figure 3 were obtained .by fixing 100 epochs and training using the stochastic gradient descent method with a learning rate of 0.1 trying to minimize the *Square-norm* error (*MSE*).

As can be seen in the Figure, almost all architectures do not approximate the sin function properly, despite the architectures *1-2-2-1* and *1-4-2-1*.
If the experiment is repeated changing the epoch number (instead of 100 change to 200) the results are almost similar (with more precision in some parts but no improvement of modelation).
However, the architectures *1-4-4-1* and *1-4-2-1* give an augment of precision significantly important.



(a) Model 1-2-4-1 (b) MSE 1-2-4-1 (c) Model 1-4-4-1 (d) MSE 1-4-4-1

Figure 4: Architectures that improves the modelisation of *sin* function

The approximation done by these architectures is so significant that allows us to observe and conclude that the model that approximates the best the sin function is $1-2-4-1$.
This conclusion is obtained because is the minimum architecture that can approximate the function, that can "learn"[5].
Moreover, it can be an evidence of the following proposal:

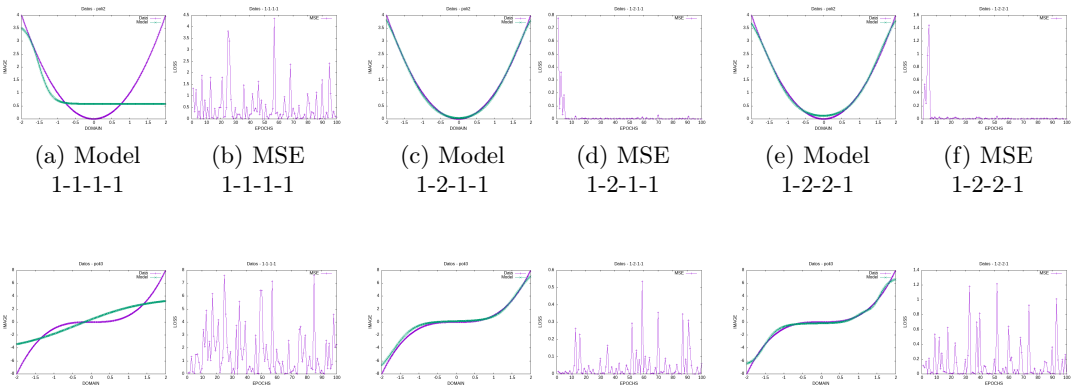
- **Proposal:** To approximate any periodical function in a real plane needs a minimum of 2 neurons in each of the $2 \cdot K$ hidden layers, where K is the sum of relatives extremes and point of inflexion of the function.

This proposal is obtained by observing that every layer of the neural network is modelling one interval (delimited by the extremes of the function and inflexion points) and performing that approximation needs at least 2 sigmoids.

2.2.2 Study of the *parabolic* function

The same experiment has been done with the function x^2 (and also the x^3 to have more results to compare) with fixed epochs, learning rate and domain.
Concretely, the values of epochs, learning rate and domain used are, respectively, 100, 0.1 and $[-2, 2]$.
The results obtained for polynomial functions are shown in 5.
It shows how the most efficient architecture (which gives the better performance as modelling as precision) is the $1-2-1-1$.
This architecture is similarly obtained before with the periodical functions but is not the same, which was one of the axioms proposed in the proposal.
Therefore, a last study will be done to obtain the architecture needed to model the irrational functions.

[5]Learn in this term refers as if the training process allows to reduce the loss (also named error) near 0 with a significant decrease throughout the epochs.



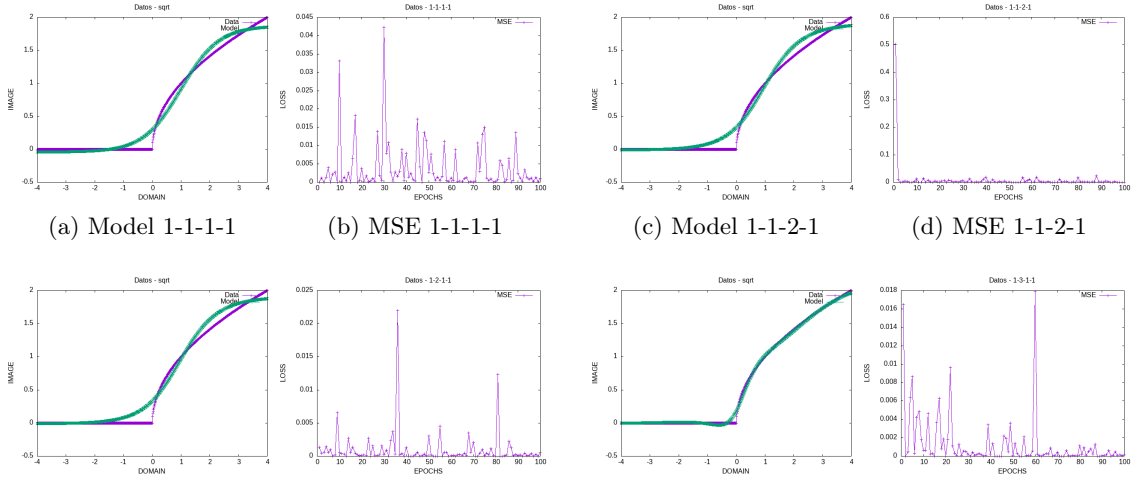
(a) Model 1-1-1-1 (b) MSE 1-1-1-1 (c) Model 1-2-1-1 (d) MSE 1-2-1-1 (e) Model 1-2-2-1 (f) MSE 1-2-2-1

(g) Model 1-1-1-1 (h) MSE 1-1-1-1 (i) Model 1-2-1-1 (j) MSE 1-2-1-1 (k) Model 1-2-2-1 (l) MSE 1-2-2-1

Figure 5: Results of different architectures approximating the functions x^2 and x^3

2.2.3 Study of the *squared root* function

Finally, reproducing the experiment for the function $\sqrt{x}$, the results obtained are shown in 6.



(a) Model 1-1-1-1 (b) MSE 1-1-1-1 (c) Model 1-1-2-1 (d) MSE 1-1-2-1

(e) Model 1-2-1-1 (f) MSE 1-2-1-1 (g) Model 1-3-1-1 (h) MSE 1-3-1-1

Figure 6: Results of different architectures approximating the function $\sqrt{x}$

In this study, the only parameter that where changed with respect to the previous experiments was the domain (which was increased to the interval $[-4, 4]$).
As it can be observed, the architecture that approximates the best the function is $1-3-1-1$.
Moreover, comment that the intuition of the inclusion of irrational functions such as the $\sqrt{x}$ in the axioms of the proposal was made to use the property of having a function similar to $ReLU$[6].
This inclusion of the function $\sqrt{(x)}$ in spite of the $ReLU$ was made thinking that was a better option and the least "forced" option of any other function that could give that

[6]The $ReLU$ function is widely used as an activation function which is defined as $f(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$

similar property. However, this text between other similar functions was not made so could be interesting to expand this study by adding and experimenting with other functions.

2.3 Viability of approximation discontinuous functions

It is important to keep in mind that splitting functions into domains can develop in discontinuities which do not allow the use of the **Universal Approximation Theorem**. Nevertheless, the application of neural networks in tasks such as classification or regression in many areas, with no information of the function that is approximating, using activation functions different from the *sigmoid* that give (apparently) correct results give the idea that possibly the theorem could be expanded and (in some cases) have no need of the continuity restriction.

- **Proposal:** If a function has discontinuities different from asymptotic, the **Universal Approximation Theorem** can be used to give an approximation with error 0 when the number of sigmoid tends to infinity.

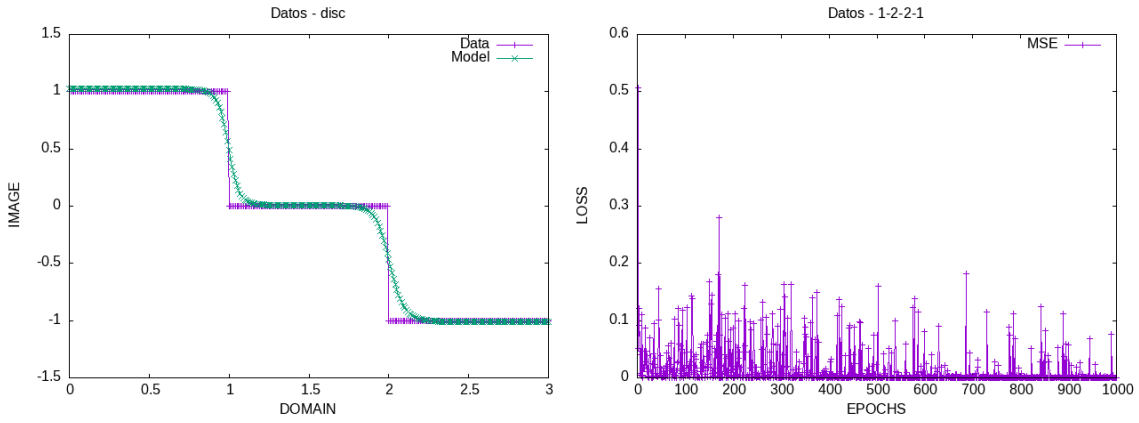
This proposal can be sustained by the ability of sigmoid functions to approximate jump discontinuities. However, an experiment with discontinuous functions has been done to give evidence for this proposal.

2.3.1 Study of Jump discontinuity

In this experiment, a simple architecture of $1-2-2-1$ with a high amount of epochs (concretely 10000) and fixed learning rate of 0.1 have been used to approximate the function:

$$f(x) = \begin{cases} 1, x \leq 1 \\ 0, x \in (1,2] \\ -1, x > 2 \end{cases}$$

The results of approximating the function in the interval $[0,3]$ are shown in 7.



(a) Model (b) MSE

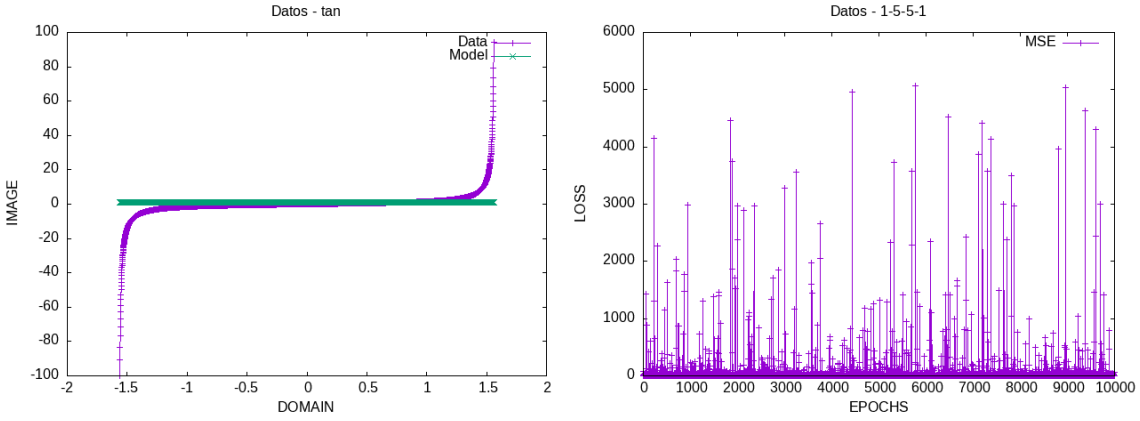
Figure 7: Architecture $1-2-2-1$ approximation to a jump discontinuity function

As can be seen in the figure 7, using a limited model with enough number of epochs the

approximation is quite acceptable. Therefore the use of *sigmoid* functions in approximate split functions with jump discontinuities can be used at least in some cases.

2.3.2 Study of Asimptotyc discontinuity

However, the application of the **Universal Approximation Theorem** in discontinuities such as asymptotic can give problems that impossibilate the modelation using sigmoids.
Repeating the experiment done with the jump discontinuity function, but with a more powerful architecture, to approximate the function $\tan x$ in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ give us an idea of this problem.
The results obtained are shown in 8.



(a) Model (b) MSE

Figure 8: Architecture $1-5-5-1$ approximation to $\tan x$ function

As can be observed the neural network is limited and cannot approximate properly the function. However, the use of other activation functions such as the *ReLU* mentioned before can give a better performance.
In the case of using the *ReLU* function, it gives another problem that can be difficult to handle, the **Universal Approximation Theorem** does not ensure that the resulting function approximates with a limited error the system.
Moreover, the use of the *BackPropagation* method and its dependency with the *Newtons Method* (the *Steepest Gradient Descend*) to obtain zeros of the *cost function* in the training process, does not ensure that the solution it arrives is the best or, even doe, if it is correct in all cases (and we can be blind in situations were the network was not tested).

2.4 Observations of the results

To sum up, all the results and conclusions obtained in the studies we can conclude if there is enough evidence for the proposal.

- **Proposal:** If there is enough information about the system to model, there exists an efficient neural network in which architecture can be obtained using rules.

To verify the viability of the proposal needed to accomplish the following conditions:

1. All periodic functions can be approximated as a combination of sinus functions.
   The first condition can be proven using *Fourier series*
   However, it is important to mention that a periodical function such as $\tan x$ can not be approximated due to the asymptotes. Therefore, a periodical function can be modelled with sin functions because it requires that can have asymptotes.

2. All polynomial functions can be approximated as a combination of linear functions.
   Such as the first condition, the second one can be proven using *Taylor Series*, not having a possible counterpart that does not apply.

3. All irrational functions can be approximated as a combination of irrational functions.
   Point 3 can be explained using the condition that we are trying to approximate a function in a certain interval. Therefore it can be obtained the following deduction:

   - **Deduction:** Approximation of irration functions.
     Being $f_n : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$ where $f_n(x) = \sqrt[n]{x}, \forall n \in \mathbb{N}/n > 1$ and $g(x) = \sqrt{x}$.

     $$\exists c, d \in \mathbb{R}/\forall x \in [a, b], f_n(x) - g(c \cdot x + d) < \epsilon$$

     Where $\epsilon \in \mathbb{R}^+$ is the error and $\{c, d\}$ are parameters that ajust g to approximate f_n in the interval $[a, b] \cup \mathbb{R}$.

   Nevertheless, mention that when n (the degree of the root) is even the interval needs to be in the set $\mathbb{R}^+$. Moreover, this deduction can handle situations when the set of functions f_n are defined as $f(x)_n = \alpha \sqrt[n]{x} + \beta$ where the parameters $\alpha, \beta \in \mathbb{R}$
   This deduction can also be demonstrated using tangency:

   - **Demostration:** Approximation of irration functions.
     Being $f_n : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$ where $f_n(x) = \sqrt[n]{x}, \forall n \in \mathbb{N}/n > 1$ and $g(x) = \sqrt{x}$.
     Being $P = (p_0, p_1)/p_0 = \frac{a+b}{2}, p_1 = f_n(p_0), \exists c, d \in \mathbb{R}/g(c \cdot p_0 + d) = p_1$. Therefore,

     $$\exists (\alpha, \beta) \subset [a, b]/p_0 = \frac{\alpha + \beta}{2} \text{ where } \forall x \in (\alpha, \beta), f_n(x) - g(c \cdot x + d) < \epsilon$$

     $\epsilon \in \mathbb{R}^+$ the error.

   These demonstrations could be used for every function, not limiting them only to irrational functions.

4. The mentioned functions can be modelled using a particular architecture.
   The studies about the different architectures of the different functions $(x^2, \sin x, \sqrt{x})$ show that there exists a differentiable minimum architecture that approximates the best of each function.

5. Exists a determined minimum number of data to train the neural network.
   That point is controversial. In all the studies the number of data available where

different with at least 200 registers. However, the relevant part about this is that it really needs a deterministic process that generates the dataset.
The minimal information that is required is the deterministic process intrinsically. If there is no deterministic process the information needed will not be enough despite how many it could recap.
This is the consequence of the backpropagation method and its dependency in the *Steepest Gradient Descend*. If there is no "correct" model to compare and interpret the functionality of the network it is not possible to "extract theoretically the best architecture".
Hence, there is no "minimum number of samples" despite there is necessary existence of a (deterministic) *non-black box*[7] method.

Concluding the observations, the studies give enough evidence of the proposal.
To give more evidence of that proof, an experimental process will be done to extract the best minimal architecture that approximates a deterministic process.
Moreover, an efficiency test will be also performed to ensure that neural networks in collaboration with the information that gives the deterministic process improve the computation time it needs.

3 Arabic to Roman numbers

The idea about that topic is the following, any natural number (normally written in the Arabic form) can be expressed as a Roman number. This process is a sequence of fixed rules that repeated steeps give the result (it is a deterministic process).
As is a deterministic function ensures the existence of a function that models the process, but this function is unknown.
The main objective is, with the observation and the proposal done before, to create a neural network that approximates this unknown function.

3.1 Preprocess of data and reduction of variables

The main problem when tracking with Roman numbers is that is a sequence of different characters that can not be handled easily. Therefore a simplification was performed.

- **Application:** Interpretation of Roman numbers.
  Being the set $C_n = \{c_0, \cdots, c_{n-1}\}$ where $\forall c \in C_n$ is a roman number.
  Let's define that each $\forall c \in C_n$ is formed by two characters: $c = (a, b)$, where b is a letter of the Roman number system, the set $\{I, V, X, C, \cdots\}$, and a the element in the roman number system just one or two positions before which is b or $\varnothing$.
  Using this assumption we can write every number, for example, the element
  $c = (I, V) \equiv 4$ and the sequence $C_3 = \{(\varnothing, X), (\varnothing, X), (I, V)\} \equiv 24$.

With this interpretation and definition of the Roman numbers set and its properties, another application can be done.

[7] A *black box* model is a process where the internal operations and functionality of it can not be interpreted partially or at all, giving the aspect of being a non-undertanable or/and verified model.

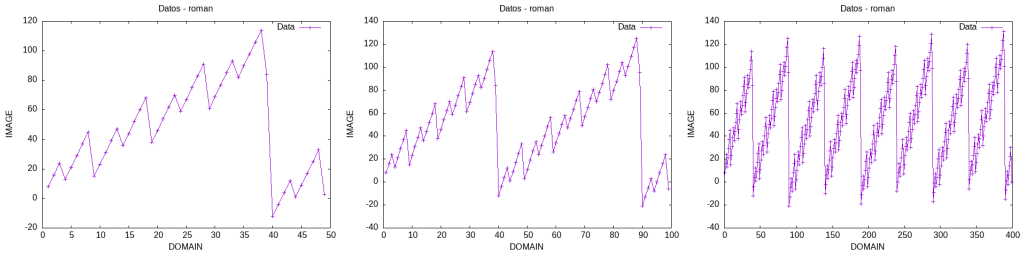
- **Application:** Reduction of liberty degrees function.
  Using the previous definition of the *Roman numbers set*, it is possible to create a function $f : \{C_n\} \to \mathbb{Z}, \forall n \in \mathbb{N}$.
  Before defining this function it essentially assigns to each letter of the Roman number system a number (for example its position in the alphabet starting with $A \equiv 1$ and defining $\varnothing \equiv 0$).
  This method allows us to represent the example sequence used before as a set of numbers:

$$C_3 = \{(\varnothing, X), (\varnothing, X), (I, V)\} \equiv \{(0, 24), (0, 24), (9, 22)\} \equiv 24$$

  Now we can create the function $f : C_n \to \mathbb{Z}, \forall n \in \mathbb{N}$ as:

$$f(C_n) = \sum_{i=0}^{n-1} b_i - a_i, \text{ where } c_i = (a_i, b_i)$$

This application allows to representation of each Roman number in one Arabic number (but does not ensure that this process satisfies the injection property).
The important about this process is that we can represent each natural number expressed in Arabic with another number (in the integers) in Arabic which is equivalent to, at least one, roman number.
In consequence, it can be represented graphically in $\mathbb{R}^2$. This representation is shown in 9.



(a) 1 to 49 (b) 1 to 99 (c) 1 to 399

Figure 9: Distribution of the associated Arabic number to the Roman natural number

The function distribution that appears is the result of applying the function defined to the deterministic rules of Roman numbers, therefore there is no knowledge of which is the function behind this process.
However is an ideal case to use the experimental observation and try to obtain with the dataset given by the deterministic method a function that approximates the distribution of the dataset.

3.2 Analisis of the dataset and creation of the neural network

The shape of the function shown in 9 tends to mimic the *Sawtooth wave* but, in this case, it is not exactly periodical. This consequence (not being periodical) creates some problems because the application of the architecture developed to approximate the periodical functions can not be applied correctly.

Despite this problem, it is possible to use the architecture developed for periodical functions (tending to approach the *Fourier Serie* of the *Sawthooth wave*) in what seems to be the period of the wave 49.
Therefore, the neural neural network will have the architecture shown in 10.

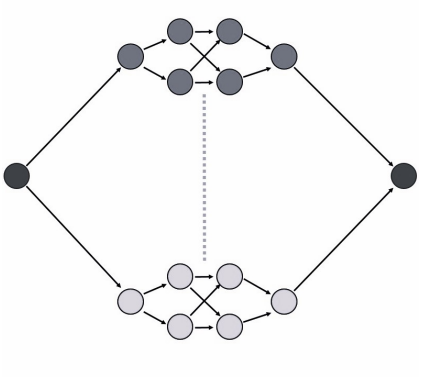
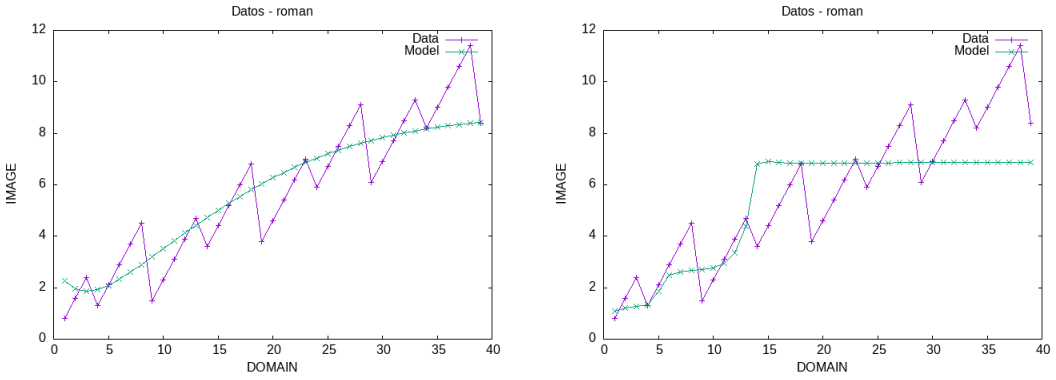


Figure 10: Proposal architecture of the neural network that approximates the Roman number distribution

This neural network will be composed of two neurons (one input and one output) that send/receive the information to/from sub-neural networks with the architecture obtained to modulate the sin function.
The neural network will be trained with only the first subset of the dataset (number between 0 and 49) and if a bigger number is introduced it only needs to be reduced as the remainder of the division of this number and the period (49).
The important part of the network is the number of sub-neural networks, the learning rate and the number of epochs that need to approximate properly the distribution.
Unfortunately, the number of epochs and learning rate can not be deduced until several have been tested.

However, the number of sub-networks could be deduced as the amount needed to approximate the *Fourier Serie* the *Sawthooth wave* with enough precision.
To test and deduce the optimal number of sub-networks and the value of the learning rate, two first approaches have been done, as can be seen in 11



(a) Network with 3 architectures $1-2-2-1$ and fixed learning rate

(b) Network with 4 architectures $1-2-2-1$ and decreasing learning rate

Figure 11: Test of different neural networks to approach the distribution of Roman numbers

The conclusion obtained from this test is that the simple architecture of the sin function of $1-2-2-1$ is not enough and could be improved by using the most robust one of $1-2-4-1$.
Moreover, the variable learning rate (decreasing its value while the epochs increase) adjusts better the pics of the distribution. However, the number of epochs testes (in the case of figure 11 1000) seems to not be enough.
Also, comment that the scale of the distribution has been decreased by one order of magnitude to reduce the variance between values and help to focus the Backpropagation when

training (by reducing the gradient vector and avoiding irrational move through the function cost reduction).

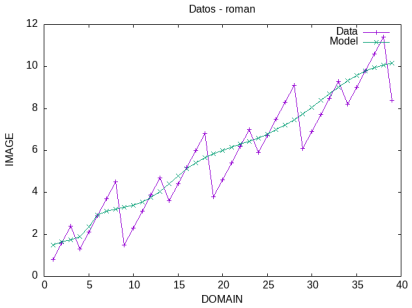


Figure 12: Improvement of the sub-network capabilities to approach the distribution of roman numbers

If the test is repeated with 4 sub-networks with architectures $1-2-4-1$, introducing the learning rate reduction by one order of magnitude each time the number of epochs done increases its order of magnitude and 10000 epochs the results obtained perform a better approach, as can be seen in 12.
From the figure, it is possible to deduce that the critical number of sub-networks has not been obtained yet. Moreover, observing the graph another conclusion can be made; the number of epochs is not enough.
Nevertheless, it is possible that the model is more complex than it should be.

Studying the distribution and behaviour of the Roman numbers, it can be deduced that it has 3 subperiods.
The first period occurs when the number ends in 4, the second when the number ends in 9 and the third period is when the number starts with 4 or 9.
In all these three cases the behaviour of the function changes (decreasing or increasing the following value much more than it would expect within its tendency). Using this information can be considered that the exact amount of subnetworks of the type $1-2-4-1$ to perform the best relation with precision-efficiency is 3 (one for each period).

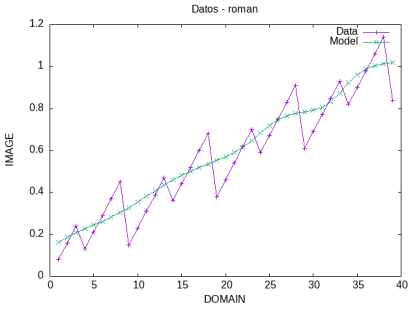


Figure 13: Final architecture proposed to approach the distribution of roman numbers

In this approach, the number of epochs has been increased to 10^6 epochs and includes 3 sub-networks with architectures $1-2-4-1$.
The results obtained in 13 are not as good as was expected but give the best precision at the moment.
However, until now there is not enough precision to consider any of these architectures a good candidate to approximate the distribution of the Roman numbers.

Nevertheless, all this unexpected non-approximation of the function could be caused by the unknown correct values of necessary epochs and/or learning rate.
This problem could not be solved using the proposal and gives problems when planning the correct architecture for modelling a process.
Therefore, as it is impossible to perform a comparison between the results of the methods due to the poor precision of the models, an analysis of training cost has to be done to verify if it is practical to make improvements in the process to allow the models to train with more capabilities (such as parallelization using GPU, improvements of the neural network code or more data an training time).

3.3 Efficiency and time-cost results

To sum up, for all four tests we can reflect results[8] in the table 1.
It can be observed that the training models used are quite good taking into account that

Sub-networks	Architecture	Epochs	Learning rate	Training time	Mean MSE
3	$1-2-2-1$	10^3	0.01	1.03"	4.09
4	$1-2-2-1$	10^3	$\{10^{-2}, 10^{-3}\}$	1.70"	2.96
4	$1-2-4-1$	10^5	$\{10^{-(2+i)}\}_{i=0}^{2}$	2'1.04"	0.87
3	$1-2-4-1$	10^6	$\{10^{-(3+2i)}\}_{i=0}^{3}$	10'17.55"	0.14

Table 1: Analisis of cost and efficiency about the different architectures develop

the epochs are "relatively low". Also, mention that the "size" of the architecture is not as big as normally MLPs are done to approach this type of problem.
Therefore it can be considered a shred of evidence that the proposal made as a set of "considerations" to develop neural networks could be verified in other dimensions and consolidated as a theorem if it is possible to prove its properties.
In the worst scenario, the study has shown that knowing the process behind a model can be used as a sustain to create neural network experts in solving this process (which could perform a fixed amount of operations to develop a task which, in its deterministic process of calculation, will not be able to determine or even have the solution in a properly amount of time).

4 Conclusions

Despite all the objectives initially planned have been achieved:

1. Develop neural networks using C language coding.
2. Propose a possible improvement of how neural networks can be designed.
3. Verify that The **Universal Approximation Theorem** definition can be expanded and used in cases that theoretically it should not be able to.
4. Propose a neural network using the obtained proposal that approximates a deterministic process.

The most challenging objective, being able to improve the time process for a deterministic process using a neural network, was not achieved.
However, during the study, it has been shown that there exists enough evidence that this challenge can be done with a better performance and analysis of the deterministic process.
It is unfair not to mention that the deterministic process selected to verify the improvements of applying neural networks could have been as easy as it has thought and with another deterministic process this verification objective could be achieved much more easily.
Also, emphasize the different proposals done during the study as can be an initial point for future studies and encourage to take it as a starting point to develop better proposals and correct any possible mistakes that could be done and not taken into consideration.

[8] the results, studies and figures that have been shown/mentioned in the report where using a Raspberry PI 4 to ensure that the computational cost and time process can be completed without any problems of different capabilities.

References

[1] Wikipedia, *Multilayer perceptron*,
https://en.wikipedia.org/wiki/Multilayer_perceptron

[2] Wikipedia, *Perceptrón*,
https://es.wikipedia.org/wiki/Perceptr%C3%B3n

[3] Wikipedia, *Activation function*,
https://en.wikipedia.org/wiki/Activation_function

[4] Wikipedia, *Frank Rosenblatt*,
https://es.wikipedia.org/wiki/Frank_Rosenblatt

[5] Wikipedia, *Activating Function*,
https://en.wikipedia.org/wiki/Activating_function

[6] Wikipedia, *Artificial neural network*,
https://en.wikipedia.org/wiki/Artificial_neural_network

[7] Wikipedia, *Neural network*,
https://en.wikipedia.org/wiki/Neural_network

[8] Wikipedia, *Universal Approximation Theorem*,
https://en.wikipedia.org/wiki/Universal_approximation_theorem

[9] Wikipedia, *Mean Squared Error*,
https://en.wikipedia.org/wiki/Mean_squared_error

[10] Wikipedia, *Mean Absolute Error*,
https://en.wikipedia.org/wiki/Mean_absolute_error

[11] Wikipedia, *Cross-Entropy*,
https://en.wikipedia.org/wiki/Cross-entropy

[12] Wikipedia, *BackPropagation*,
https://en.wikipedia.org/wiki/Backpropagation

[13] Dantzig, G.B. y P. Wolfe, «Decomposition principle for linear programs», *Operations Research*, **8**, págs. 101–111, 1960.