An Analytical Approach to Compute the Exact Preimage of Feed-Forward Neural Networks

Théo Nancy, Vassili Maillet, Johann Barbier

March 3, 2022

Abstract

Neural networks are a convenient way to automatically fit functions that are too complex to be described by hand. The downside of this approach is that it leads to build a black-box without understanding what happened inside. Finding the preimage would help to better understand how and why such neural networks had given such outputs. Because most of the neural networks are noninjective function, it is often impossible to compute it entirely only by a numerical way. The point of this study is to give a method to compute the exact preimage of any Feed-Forward Neural Network with linear or piecewise linear activation functions for hidden layers. In contrast to other methods, this one is not returning a unique solution for a unique output but returns analytically the entire and exact preimage.

Notations and assumptions

Notations

Numbers and Arrays

a, A	A scalar (real or integer)
$\mathbf{a}$	A vector
$\mathbf{A}$	A matrix
$\mathbf{I}_n$	The identity matrix of size n
$\mathbf{0}_{n,m}$	The null matrix of dimensions $n \times m$

Indexing

a_i	Element i of a vector $\mathbf{a}$, with indexing starting at 1

Sets

$\mathbb{A}$	A set
$[a, b]$	The real interval including a and b
$[a, b[$	The real interval including a but excluding b
$[\![n, m]\!]$	The integer interval including n and m
$[\![n, m[\![$	The integer interval including n but excluding m

Functions

$f : \mathbb{A} \longrightarrow \mathbb{B}$	The function f with domain $\mathbb{A}$ and codomain $\mathbb{B}$
f^{-1}	The preimage function of f
sgn	The sign function
Id	The identity function
$ReLU$	The Rectified Linear Unit function
$PReLU$	The Parametric Rectified Linear Unit function

Abbreviations

ANN	Artificial Neural Network
$FFNN$	Feed Fordward Neural Network

Assumptions

The Lesbegue measure of the set of singular matrices is zero. Note that the probability for a random matrix to be singular is not zero on a computer because of their finite precision. However it remains very unlikely for a random squared matrix to be singular in theory. As a result we will assume that all squared matrices in this study are invertible.

Contents

1 Introduction

The solutions of some problems are easy to explain by giving multiple examples but tremendously hard to describe analytically. This is for example the case of image recognition. The difference between a dog and a cat picture is instantly discovered by a human eye. It is however almost impossible to mathematically explain what is a dog and what is a cat. Since AlexNet in 2012 [5], ANN has been shown as a wonderful way for computer scientists to solve complex machine learning problems [8]. Thanks to repeated training on examples, ANN can learn to solve such complex problems as image recognition by fitting a large function that describe the best the given problem depending on the given examples.
According to the universal approximation theorem, ReLU networks with $n + 1$ layers can approximate any continuous function of n-dimensional input variables [3]. It means that every problem were the solution can be expressed as a continuous function can theoretically be solved by an ANN.
If the ANN correctly approximates the function, most of the information about of the problem has been solved would be contained inside the ANN weights. One of the major issues that encounter ANN are their lack of comprehensibility. A way to better understand solutions given by neural networks would be to know how the inputs are reacting according to the outputs. The perfect understanding of the relationship between outputs and inputs is provided by the preimage which gives all corresponding inputs for given outputs. The issue that remains is how to compute it.

Finding inputs for any neural network's outputs has already been done by giving one approximate and unique solution from the entire set of solutions [2]. The most natural method to approximate inverse images of a FFNN output is by retro-propagating it on inputs using gradient descent while freezing all biases and weights. More recently, researchers built neural networks in order to invert other neural networks for microwave design filter application [4]. However, it still doesn't give the entire set of solutions which leaves the inverse modeling problem unsolved. Note that these solutions can sometimes not be used in practice because of constraints on inputs. The problem which remains is the nonuniqueness of the preimage which has an infinite number of solutions when the number of inputs is strictly higher than the number of outputs. It leads every current method to return a unique approximation of a preimage solution for a given output, even if the proposed solution is not compatible with the problem constraints. As a consequence, because many ANN have an infinity of corresponding inputs for their outputs, it is impossible to approximate the preimage numerically.
The goal that remains is to find the preimage of the ANN which would give the entire set of solutions for any outputs. In order to compute a complete preimage, it has to be done analytically . We found two articles which explored this subject. The first focuses on finding the preimage of a single point [1] while the second extends it to compute the preimage of polytopes [6].

This study aims at giving a method to analytically find the exact preimage of any given FFNN where activation of hidden layers are linear or piecewise linear functions like ReLU or Leaky ReLU function. Note that every problem that can be modeled by a continuous function can be approximated with a piecewise linear function like ReLU or Parametric ReLU [3]. The point is to set up the basis of ANN preimage computing by giving the details of a fully working and simple method. The first part of the study focuses on how to analytically solve this preimage problem in a linear single layer case. The second shows how to iterate it to give a solution for any linear multilayers network. The third part focuses on extending to piecewise linear cases and splitting them to linear problems. The last part is a description of the implied space and time complexity.
It will be shown that because of its exponential time complexity, the exact analytical case can't be used for large neural networks in practice. However, the sets of solutions can be voluntarily reduced in order to compute it faster and made the algorithm usable in a practical case.

2 Linear cases

2.1 Determined system

Let's consider a neural network with only two layers : input and output layer with the same number of neurons for both of them.

Let $N \in \mathbb{N}^*$, the number of neurons.
Let $\mathbf{W} \in \mathbb{R}^{N \times N}$, the matrix of weights.
Let $\mathbf{x}, \mathbf{y}, \mathbf{b} \in \mathbb{R}^N$, respectively inputs, outputs and bias.
Let $\alpha, \beta \in \mathbb{R}$
Let $f : x \mapsto \alpha x + \beta$, the activation of the layer.
An output can be expressed as follows :

$$y_k = f(\sum_{i=1}^{N} w_{k,i} x_i + b_k), k \in [\![1, N]\!]$$

In the case where the activation function is linear, the previous equation can simply be rewrite as follow :

$$y_k = (\sum_{i=1}^{N} \alpha w_{k,i} x_i + \alpha b_k + \frac{\beta}{N}), k \in [\![1, N]\!] \tag{1}$$

$\forall k, i \in [\![1, N]\!], w'_{k,i} = \alpha w_{k,i}, b'_k = \alpha b_k + \frac{\beta}{N}$

$$y_k = (\sum_{i=1}^{N} w'_{k,i} x_i + b'_k), k \in [\![1, N]\!]$$

It means that finding an inverse image of a layer can be reduced as a matrix problem :

$$\mathbf{W'x} = \mathbf{y} - \mathbf{b'}, \mathbf{W'} = \begin{bmatrix} w'_{1,1} & w'_{1,2} & \cdots & w'_{1,N} \\ w'_{2,1} & w'_{2,2} & \cdots & w'_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ w'_{N,1} & w'_{N,2} & \cdots & w'_{N,N} \end{bmatrix} \tag{2}$$

This can be easily solved with Gauss elimination in $O(N^3)$ time complexity for the following system :

$$\mathbf{S} = \left[\begin{array}{cccc|c} w'_{1,1} & w'_{1,2} & \cdots & w'_{1,N} & y_1 - b'_1 \\ w'_{2,1} & w'_{2,2} & \cdots & w'_{2,N} & y_2 - b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w'_{N,1} & w'_{N,2} & \cdots & w'_{N,N} & y_N - b'_N \end{array} \right]$$

Solving this system will lead to build a function P of dimension N which gives a solution for any output with no need to recall the algorithm. This function P will be expressed as:

$$P(y_1, ..., y_N) = \begin{bmatrix} (\sum_{i=1}^{N} a_{i,1} y_i) + c_1 \\ \vdots \\ (\sum_{i=1}^{N} a_{i,N} y_i) + c_N \end{bmatrix}$$

Where the $a_{i,j}$ and c_j are real values and P the exact preimage of $\mathbf{S}$.
P can also be written as a matrix product:

$$P(\mathbf{y}) = \mathbf{A}\mathbf{y} + \mathbf{c}$$

Where $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{c} \in \mathbb{R}^{N}$.

2.2 Overdetermined systems

Now let's consider a neural network once again with only inputs and outputs layers, where the input layer has N neurons and outputs layer M neurons where N is strictly higher than M. As previously seen, if the activation is linear the layer can be inverted by writing the problem as if the activation was the identity function. In order to simplify the notations, the next calculations will be using the identity function.

Now (1) and (2) can be rewritten as follows :

Let $N, M \in \mathbb{N}^*, N > M$

$$y_k = \sum_{i=1}^{N} w_{k,i} x_i + b_k, k \in [\![1, M]\!] \tag{3}$$

$$\mathbf{W}\mathbf{x} = \mathbf{y} - \mathbf{b}, \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,N} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,N} \end{bmatrix} \tag{4}$$

Even with more unknown inputs than outputs , it is still possible to find a solution, writing it as a set of dimension $N - M$, considering $N - M$ inputs as free variables for M dependent variables.

As previously, the system can be written a follows :

$$\mathbf{S} = \left[\begin{array}{cccc|c} w_{1,1} & w_{1,2} & \cdots & w_{1,N} & y_1 - b_1 \\ w_{2,1} & w_{2,2} & \cdots & w_{2,N} & y_2 - b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,N} & y_M - b_M \end{array}\right]$$

In order to solve this system by Gaussian elimination, $N-M$ free variables must be moved to the right member as **S** is overdetermined.

Let $\mathbf{W}_1 = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,M} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,M} \end{bmatrix}$,$\mathbf{W}_2 = \begin{bmatrix} w_{1,M+1} & w_{1,M+2} & \cdots & w_{1,N} \\ w_{2,M+1} & w_{2,M+2} & \cdots & w_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,M+1} & w_{M,M+2} & \cdots & w_{M,N} \end{bmatrix}$

Let $\mathbf{x}_1 = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}$, $\mathbf{x}_2 = \begin{bmatrix} x_{M+1} \\ \vdots \\ x_N \end{bmatrix}$

$$\begin{aligned} \mathbf{Wx} = \mathbf{y} - \mathbf{b} &\Rightarrow \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \mathbf{y} - \mathbf{b} \\ &\Rightarrow \mathbf{W}_1\mathbf{x}_1 + \mathbf{W}_2\mathbf{x}_2 = \mathbf{y} - \mathbf{b} \\ &\Rightarrow \mathbf{W}_1\mathbf{x}_1 = \mathbf{y} - \mathbf{b} - \mathbf{W}_2\mathbf{x}_2 \end{aligned}$$

The previous system can be rewritten as :

$$\textbf{S'} = \left[\begin{array}{cccc|c} w_{1,1} & w_{1,2} & \cdots & w_{1,M} & y_1 - b_1 - \sum_{i=M+1}^{N} w_{1,i}x_i \\ w_{2,1} & w_{2,2} & \cdots & w_{2,M} & y_2 - b_2 - \sum_{i=M+1}^{N} w_{2,i}x_i \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,M} & y_M - b_M - \sum_{i=M+1}^{N} w_{M,i}x_i \end{array} \right]$$

As **S'** is now a determined system it can be solved by Gaussian elimination. The preimage will be a function P of dimension N that depends on the $y_{i \in \llbracket 1,M \rrbracket}$ and also on the new real free variables $\tau_i \in \llbracket 1, N-M \rrbracket$.
The τ_i belong to the vector $\boldsymbol{\tau} \in \mathbb{R}^{N-M}$.

$$\begin{aligned} P(y_1, ..., y_M) &= \begin{bmatrix} (\sum_{i=1}^{M} u_{1,i}y_i) + (\sum_{i=1}^{N-M} v_{1,i}\tau_i) + c_1 \\ \vdots \\ (\sum_{i=1}^{M} u_{N,i}y_i) + (\sum_{i=1}^{N-M} v_{N,i}\tau_i) + c_N \end{bmatrix} \\ &\Rightarrow P(\mathbf{y}) = \mathbf{Uy} + \mathbf{V}\boldsymbol{\tau} + \mathbf{c} \end{aligned}$$

Where $\mathbf{U} \in \mathbb{R}^{N \times M}$,$\mathbf{V} \in \mathbb{R}^{N \times (N-M)}$ and $\mathbf{c} \in \mathbb{R}^{N}$.

2.3 Underdetermined systems

In the case where M is strictly higher than N , the system is underdetermined. Let **X** the space of inputs and **Y** the space of outputs.

$$\begin{aligned} f(X) \subset Y &\Rightarrow dim f(X) + dim\{x, f(x) = b\} = dim X \\ &\Rightarrow dim f(X) \leq dim X = N < M \end{aligned}$$

If the system have F free variables where F is higher than $M - N$ and no hidden layers with strictly less neurons than the outputs layer then the system can be solved as follows :

Let Γ be the space of outputs.
To exactly solve an underdetermined case, Γ has to be expressed as a linear combination of y and other free variables in order to make the system determined.
Let $N, M \in \mathbb{N}^*, N < M$
Let $F > M - N$
Let $\boldsymbol{\tau} \in \mathbb{R}^F$, a vector of F free variables that the y_k will depend on.
Let $\mathbf{A} \in \mathbb{R}^{M \times F}$.
Let $\mathbf{y}, \boldsymbol{\gamma}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^M$.

$$\mathbf{S} = \left[\begin{array}{cccc|c} w_{1,1} & w_{1,2} & \cdots & w_{1,N} & \gamma_1 - b_1 \\ w_{2,1} & w_{2,2} & \cdots & w_{2,N} & \gamma_2 - b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,N} & \gamma_M - b_M \end{array}\right] \tag{5}$$

Where $\gamma_k = (\sum_{i=1}^{F} a_{k,i}\tau_i) + y_k + c_k; k \in [\![1, M]\!]; a_{k,i}, c_k \in \mathbb{R}$

Let $\mathbf{A}_1 = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,M-N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,M-N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & a_{M,2} & \cdots & a_{M,M-N} \end{bmatrix}$, $\mathbf{A}_2 = \begin{bmatrix} a_{1,M-N+1} & a_{1,M-N+2} & \cdots & a_{1,F} \\ a_{2,M-N+1} & a_{2,M-N+2} & \cdots & a_{2,F} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M,M-N+1} & a_{M,M-N+2} & \cdots & a_{M,F} \end{bmatrix}$

Let $\boldsymbol{\tau}_1 = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_{M-N} \end{bmatrix}$, $\boldsymbol{\tau}_2 = \begin{bmatrix} \tau_{M-N+1} \\ \vdots \\ \tau_F \end{bmatrix}$

Let $\boldsymbol{\gamma}_1 = \mathbf{A}_1\boldsymbol{\tau}_1, \boldsymbol{\gamma}_2 = \mathbf{A}_2\boldsymbol{\tau}_2$

$$\begin{aligned} \mathbf{W}\mathbf{x} = \boldsymbol{\Gamma} - \mathbf{b} &\Rightarrow \mathbf{W}\mathbf{x} = \boldsymbol{\gamma}_1 + \boldsymbol{\gamma}_2 + \mathbf{y} + \mathbf{c} - \mathbf{b} \\ &\Rightarrow \mathbf{W}\mathbf{x} - \boldsymbol{\gamma}_1 = \boldsymbol{\gamma}_2 + \mathbf{y} + \mathbf{c} - \mathbf{b} \\ &\Rightarrow \mathbf{W}\mathbf{x} - \mathbf{A}_1\boldsymbol{\tau}_1 = \boldsymbol{\gamma}_2 + \mathbf{y} + \mathbf{c} - \mathbf{b} \\ &\Rightarrow \begin{bmatrix} \mathbf{W} & \mathbf{A}_1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\tau}_2 \end{bmatrix} = \boldsymbol{\gamma}_2 + \mathbf{y} + \mathbf{c} - \mathbf{b} \end{aligned} \tag{6}$$

Let $\mathbf{W}' = \begin{bmatrix} \mathbf{W} & \mathbf{A}_1 \end{bmatrix}, \mathbf{x}' = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\tau}_2 \end{bmatrix}, \gamma' = \boldsymbol{\gamma}_2 + \mathbf{y} + \mathbf{c}$

$$(6) \Rightarrow \mathbf{W}'\mathbf{x}' = \boldsymbol{\gamma}' - \mathbf{b} \tag{7}$$

The system (5) can now be rewritten by adding $M - N$ variables of $\boldsymbol{\tau}$ as unknown variables of (7).
Once again, the system (7) can be solved thanks to Gaussian elimination with M solutions of $D = F - M + N$ dimensions. P will have the same form as the determined case.

2.4 Underdetermined system without extra free variables

A specific case occurs when the dimension of implied output space is strictly higher than input space. In fact, when the output layer has strictly more neurons than the input layer, most of the target outputs would be not valid and will have no solution in the preimage set. However, it is still possible to find the closest valid output in terms of least squares error in order to perform the previously shown algorithm to find a corresponding input. The following section will quickly shows how to applied least squared error to solve underdetermined systems.

Let $N, M \in \mathbb{N}^*, N < M$, respectively the inputs and outputs length.
Let $\mathbf{W} \in \mathbb{R}^{M\times N}$.
Let $\mathbf{y} \in \mathbb{R}^{M}$, the outputs vector.
Let $\mathbf{x} \in \mathbb{R}^{N}$, the inputs vector.

$$\mathbf{W}\mathbf{x} = \mathbf{y}$$

This system can't be solved for $\mathbf{x}$ in a general case. However it is still possible to find the vector $\widehat{\mathbf{y}}$ that minimizes the least squares criterion for $\mathbf{y}$. Then the system can be solved by simple Gaussian Elimination and will returns the closest solution in terms of least squares error.

Let $\boldsymbol{\varepsilon} \in \mathbb{R}^{M}, \widehat{\mathbf{y}} = \mathbf{y} + \boldsymbol{\varepsilon}$, the error vector.
The goal is now the find the smallest corresponding $\boldsymbol{\varepsilon}$ in terms of $L2$-norm.
Let rewrite the system such as :

$$\mathbf{W}\mathbf{x} = \widehat{\mathbf{y}} = \mathbf{y} + \boldsymbol{\varepsilon}$$

Because M is strictly higher than N, there are $M - N$ missing variables to keep the system determined. These $M - N$ variables can be taken from $\boldsymbol{\varepsilon}$ and treated as unknown.
Let $\boldsymbol{\varepsilon}_1 \in \mathbb{R}^{M-N}, \boldsymbol{\varepsilon}_2 \in \mathbb{R}^{N}, \boldsymbol{\varepsilon} = \begin{bmatrix}\boldsymbol{\varepsilon}_1 & \boldsymbol{\varepsilon}_2\end{bmatrix}^T$

$$\mathbf{W}\mathbf{x} = \mathbf{y} + \begin{bmatrix}\boldsymbol{\varepsilon}_1 & \boldsymbol{\varepsilon}_2\end{bmatrix}^T \Rightarrow \begin{bmatrix}\mathbf{W} & \begin{matrix}-\mathbf{I}_{M-N} \\ \mathbf{0}_{N,M-N}\end{matrix}\end{bmatrix}\mathbf{x} = \mathbf{y} + \begin{bmatrix}\mathbf{0}_{1,N} & \boldsymbol{\varepsilon}_2\end{bmatrix}^T$$

Under this form the system becomes determined and can be solved by Gaussian Elimination. The two vector $\mathbf{x}$ and $\boldsymbol{\varepsilon}_1$ will be expressed as linear combination of $\mathbf{y}$ and $\boldsymbol{\varepsilon}_2$.
Because, $\widehat{\mathbf{y}}$ is the closest valid output of $\mathbf{y}$ in terms of least squares, the $L2$-norm of $\boldsymbol{\varepsilon}$ have to be minimized.

Let $\mathbf{A} \in \mathbb{R}^{(M-N)\times M}, \mathbf{B} \in \mathbb{R}^{(M-N)\times N}, \boldsymbol{\varepsilon}_1 = \mathbf{A}\mathbf{y} + \mathbf{B}\boldsymbol{\varepsilon}_2$.

$$\begin{aligned}
\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} &= \begin{bmatrix}\boldsymbol{\varepsilon}_1^T & \boldsymbol{\varepsilon}_2^T\end{bmatrix}\begin{bmatrix}\boldsymbol{\varepsilon}_1\\ \boldsymbol{\varepsilon}_2\end{bmatrix}\\
&= \boldsymbol{\varepsilon}_1^T\boldsymbol{\varepsilon}_1 + \boldsymbol{\varepsilon}_2^T\boldsymbol{\varepsilon}_2 + 2\boldsymbol{\varepsilon}_2^T\boldsymbol{\varepsilon}_1\\
&= \left[\mathbf{A}\mathbf{y} + \mathbf{B}\boldsymbol{\varepsilon}_2\right]^T\left[\mathbf{A}\mathbf{y} + \mathbf{B}\boldsymbol{\varepsilon}_2\right] + \boldsymbol{\varepsilon}_2^T\boldsymbol{\varepsilon}_2 + 2\boldsymbol{\varepsilon}_2^T\left[\mathbf{A}\mathbf{y} + \mathbf{B}\boldsymbol{\varepsilon}_2\right]\\
&= \left[\mathbf{A}\mathbf{y} + (\mathbf{B} + \mathbf{I}_n)\boldsymbol{\varepsilon}_2\right]^T\left[\mathbf{A}\mathbf{y} + (\mathbf{B} + \mathbf{I}_n)\boldsymbol{\varepsilon}_2\right]
\end{aligned}$$

The next step is to find the corresponding $\boldsymbol{\varepsilon}_2$ that minimize the previous expression.
Let $f : \mathbb{R}^N \longrightarrow \mathbb{R}, f(\mathbf{x}) = \left[\mathbf{A}\mathbf{y} + (\mathbf{B} + \mathbf{I}_n)\mathbf{x}\right]^T\left[\mathbf{A}\mathbf{y} + (\mathbf{B} + \mathbf{I}_n)\mathbf{x}\right]$

$$\begin{aligned}
&\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) = 2\left[\mathbf{A}\mathbf{y} + (\mathbf{B} + \mathbf{I}_n)\mathbf{x}\right]^T\left[\mathbf{B} + \mathbf{I}_n\right]\\
&\frac{\partial^2 f}{\partial \mathbf{x}^2}(\mathbf{x}) = 2\left[\mathbf{B} + \mathbf{I}_n\right]^T\left[\mathbf{B} + \mathbf{I}_n\right] > 0
\end{aligned}$$

Because the second derivative of f is positive, finding the $\mathbf{x}$ that minimizes it is equivalent to find thezeros of its derivative.

$$\begin{aligned}
\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}) = 0 &\Rightarrow 2\left[\mathbf{A}\mathbf{y} + (\mathbf{B} + \mathbf{I}_n)\mathbf{x}\right]^T\left[\mathbf{B} + \mathbf{I}_n\right] = 0\\
&\Rightarrow (\mathbf{B} + \mathbf{I}_n)\mathbf{x} = -\mathbf{A}\mathbf{y}
\end{aligned}$$

The equation can be solved under this form thanks to Gaussian Elimination.

2.5 Multilayers network

Inverting a multilayer network only needs to iterate the previous method on each layer to find the exact preimage. The inputs at each iteration will be the previous outputs.
This comes from the fact that for multivariate functions of $\mathbb{R}$, the preimage of the composition is the composition of the preimages [6].

3 Piecewise linear cases

The previous section shows how to compute the exact preimage of a multilayer FFNN with linear activation. The same problem can be solved in case of nonlinear activation under particular constraints.

Let f a nonlinear activation function which is bijective in at least one interval I.
Let f^{-1} the inverse function of f in I.
Let $N, M \in \mathbb{N}^*$.
Considering the same notations as previously :

$$y_k = f(\sum_{i=1}^{N} w_{k,i}x_i + b_k), k \in [\![1, M]\!] \Rightarrow f^{-1}(y_k) = \sum_{i=1}^{N} w_{k,i}x_i + b_k, k \in [\![1, M]\!]$$

The system $\mathbf{S}$ describes the inverse images set of any layer in a FFNN where there is more neurons in the input layer than in the output layer :

$$\mathbf{S} = \left[\begin{array}{cccc|c} w_{1,1} & w_{1,2} & \cdots & w_{1,N} & f^{-1}(y_{1)} - b_1 \\ w_{2,1} & w_{2,2} & \cdots & w_{2,N} & f^{-1}(y_2) - b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,N} & f^{-1}(y_M) - b_M \end{array}\right] \tag{8}$$

This system cannot be solved in the general case where f^{-1} is nonlinear and $dim(y_i) \geqslant 1, i \in [\![1, M]\!]$. However, when $y_i, f^{-1}(y_i) \in \mathbb{R}$, the system (8) can be solved by simple Gaussian elimination. It means that any function f where $f^{-1} : \mathbb{R} \longrightarrow \mathbb{R}$ exists can be used as the output activation of a network without preventing it from being inverted.

3.1 Piecewise linear activation : Parametric ReLU

Specific nonlinear cases can be solved if the activation function is piecewise linear.

As a reminder :

$$PReLU : \mathbb{R} \longrightarrow \mathbb{R}$$

$$PReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

Let's now consider the inverse function of PReLU:

Let $\alpha \geqslant 0$
Let $iPRL : \mathbb{R} \longrightarrow \mathbb{R}$, the inverse function of Parametric ReLU of parameter α.

$$iPRL(y) = \begin{cases} y, & \text{if } y > 0 \\ \frac{y}{\alpha}, & \text{otherwise} \end{cases}$$

The only thing which changes from the linear case is that the truth value of $y > 0$ must be known. The answer is trivial if y is known. If y is expressed as a linear combination of free variables from previous equations, this problem can be reduced as a linear inequalities system :

$$\begin{cases} (\sum_{i=1}^{N} w_{i,1}x_i) + b_1 \geqslant 0 \\ \\ (\sum_{i=1}^{N} w_{i,2}x_i) + b_2 \geqslant 0 \\ \vdots \\ (\sum_{i=1}^{N} w_{i,M}x_i) + b_M \geqslant 0 \end{cases} \Rightarrow \mathbf{WX} + \mathbf{b} \geqslant 0 \tag{9}$$

Notice that there are 2^M different systems to solve depending on whether $y_j > 0$ or $y_j < 0$. In this study, these systems were all solved independently. Also notice that these systems can all be written with left member higher than 0 by multiplying it by -1 which will reverse the sign of the inequation as an effect.

This previous section shows that the method will work with ReLU activation if any system of multivariate linear inequalities can be analytically solved. The next section explains how to solve such systems.

3.2 Solving systems of multivariates linear inequalities

The goal of this section is to analytically solve multivariate linear inequalities systems.

Expression of the problem

Let $N, M \in \mathbb{N}^*$
Let $\{w_{i,j}\}_{i\in[\![1,N]\!],j\in[\![1,M]\!]} \in \mathbb{R}^{N+M}$, a family of real constants.
Let $\{b_j\}_{j\in[\![1,M]\!]} \in \mathbb{R}^M$, another family of real constants.
Let $k \in [\![1, M]\!]$

In a general case, inequalities can be written with the following form :

$$(\sum_{i=1}^{N} w_{i,k}x_i) + b_k \geqslant 0 \tag{10}$$

Considering (10), a system of multivariates linear inequalities can be described as follows in the general case:

$$S = \begin{cases} (\sum_{i=1}^{N} w_{i,1}x_i) + b_1 \geqslant 0 \\ (\sum_{i=1}^{N} w_{i,2}x_i) + b_2 \geqslant 0 \\ \vdots \\ (\sum_{i=1}^{N} w_{i,M}x_i) + b_M \geqslant 0 \end{cases} \tag{11}$$

Note that large inequalities can be replaced by strict inequalities without changing the method.

Analytical Algorithms for solving linear inequalities systems

In general case, linear inequalities system can be written as:

$$\mathbf{Wx} \geqslant 0$$

A classic algorithm to analytically solve this kind of system is Fourier-Motzkin Elimination [7]. This algorithm is simple but has the downside of being extremely slow in terms of time complexity because of adding multiple redundant inequalities. However, from a theoretical point of view, such an algorithm still provides a constructivist way to analytically solve linear inequalities systems.
The details of Fourier-Motzkin Elimination algorithm is given in Appendix A.

Solutions

The analytical solutions of a multivariate linear inequalities system solve by Fourier-Motzkin Elimination or any other analytical algorithm will looks as follows :

Let $N, M \in \mathbb{N}^*$
Let $\mathbf{W} \in \mathbb{R}^{N \times M}$ a squared matrix of real constants.
Let $\mathbf{x} \in \mathbb{R}^{M}$ a vector of real variables.
Let $\mathbf{S}$ the solution of the following system of multivariate linear inequalities :

$$\mathbf{Wx} \geqslant 0$$

Let $j_1, j_2 \in [\![0, M]\!], M = j_1 + j_2$, respectively the number of constrained and unconstrained variables.
Let $\{L_i\}_{i \in [\![1, j_1]\!]}, \{G_i\}_{i \in [\![1, j_1]\!]}$ respectively the sets of constraints that have to be lower and greater than a given x_i.

$$\mathbf{S} = \{x_i\}_{i \in [\![1,M]\!]}, \begin{cases} min(L_1) \leqslant x_1 \leqslant max(G_1) \\ \vdots \\ min(L_{j_1}) \leqslant x_{j_1} \leqslant max(G_{j_1}) \\ -\infty < x_{j_1+1} < +\infty \\ \vdots \\ -\infty < x_{j_1+j_2} < +\infty \end{cases}$$
$$= \mathbf{x}, \begin{cases} min(\mathbf{L}) \leqslant \mathbf{x_1} \leqslant max(\mathbf{G}) \\ -\infty < \mathbf{x_2} < +\infty \end{cases}$$

Let $i \in [\![1, j_1]\!]$ such as L_i, G_i are the i-th component of $\mathbf{L}, \mathbf{G}$.
Let $n \in \mathbb{N}$, number of constraints for L_i.
Let $\mathbf{A} \in \mathbb{R}^{n \times M}$, the coefficient matrix of the constraints variables in L_i.
Let $\mathbf{B} \in \mathbb{R}^{n \times M}$, the coefficient matrix of the unconstraints variables in L_i.
$\forall h \in [\![1, n]\!], \forall k \in [\![1, M]\!], k \geqslant j_1, a_{h,k} = 0$.
Respectively, $\forall h \in [\![1, n]\!], \forall k \in [\![1, M]\!], k > j_2, b_{h,k} = 0$.
Let $\mathbf{c} \in \mathbb{R}^n$, a vector of real constants.
The expression of G_i will be symmetrical to L_i.

$$L_i = \left[\sum_{k=1}^{i-1} a_{1,k}x_k + \sum_{k=1}^{j_2} b_{1,k}x_k + c_1 \quad \cdots \quad \sum_{k=1}^{i-1} a_{n,k}x_k + \sum_{k=1}^{j_2} b_{n,k}x_k + c_n\right]^T$$
$$= (\mathbf{A} + \mathbf{B})\mathbf{x} + \mathbf{c}$$

As seen previously, it is possible to solve any systems of multivariate linear inequalities analytically. It means that the last step to reverse the layer is to compute each solution by solving each 2^M different system. After that, the next layer can be inverse thanks to the same method. Notice that this time, variables would be under constraints, but it doesn't change the method as these constraints can be added as new inequalities in the next system to solve.

3.3 Application to piecewise linear case

Let $N, M \in \mathbb{N}^*$, the number of inputs and outputs neurons.
Let $\mathbf{W} \in \mathbb{R}^{N \times M}$, the matrix of weights.
Let $\mathbf{x}, \mathbf{b} \in \mathbb{R}^N$, respectively inputs and bias.
Let $\mathbf{y} \in \mathbb{R}^M$, the outputs where $\mathbf{y}$ is a vector of N variables (free or under constraints).

The preimage of the network can be computed by iteratively solve the following equation for each layer :

$$\mathbf{Wx} = \mathbf{y} - \mathbf{b}$$

Note that because the $y_{i \in [\![1,N]\!]}$ are variables, they have to be kept as symbols during the entire computation. Thanks to the previous result, multivariate inequalities systems can be solved in a general case. It means that for each layer of size $k \in \mathbb{N}^*$, the 2^k different inequalities systems can be analytically solved. Solving these systems will lead to create 2^k new sets of solutions that will have to be taken as new starting points for the next iteration of layer preimage finding.

Example:
Let's consider a neural network with piecewise linear activation for each hidden layers.
Let $N, M \in \mathbb{N}^*, M \geqslant N$, respectively the length of the inputs and outputs layer.
Let $H \in \mathbb{N}$, the number of hidden layers.
Let $\{N_k\}_{k \in [\![1,H]\!]} \in \mathbb{N}^H$, the family that contains the length of each k-th hidden layers.

As previously seen,the k-th hidden layer introduces 2^{N_k} inequalities systems to solve which lead to create 2^{N_k} new sets of solutions with their constraints. By direct induction, the total number of solution sets for the neural network Ω will be expressed as :

$$\Omega = 2^{N + \sum_{k=1}^{H} N_k}$$

Knowing this, the last step is to express the form of the preimage.
The $P_{k \in [\![1,\Omega]\!]}$ solutions sets of $\mathbf{P}$ can be written as follows:

Let $\mathbf{W} \in \mathbb{R}^{M \times N \times \Omega}$, the matrix of coefficients for the y.
Let $\mathbf{c} \in \mathbb{R}^{N \times \Omega}$, the constants coefficients.
Let $\mathbf{y} \in \mathbb{R}^M$, the outputs vector
The preimage $\mathbf{P}$ will depend of the values of the $y_{i \in [\![1,N]\!]}$ and will have constraints given by the $\tau_{j \in [\![1,M-N]\!]}$.

Let $T' \in \mathbb{N}$, the number of variables τ that were eliminated during the whole computation of the preimage.
Let $T = M - N - T'$, the number of τ that still has free variables.
Let $\mathbf{A} \in \mathbb{R}^{T \times N \times \Omega}$, the coefficients matrix for the τ.
Let $\boldsymbol{\tau} \in \mathbb{R}^{T \times \Omega}$, the extra free variable. Let $\boldsymbol{\tau'} \in \mathbb{R}^{T' \times \Omega}$, the vector of the eliminated τ that became constants real constraints of the solutions.
Let $\mathbf{L}, \mathbf{G}, \mathbf{L'}, \mathbf{G'}$ respectively the sets of constraints that have to be lower and greater than a given τ and τ'.

Let $k \in [\![1, \Omega]\!]$, the index of the solution.
$\mathbf{P}$ is a set of dimension Ω per N.
The k-th solution of $\mathbf{P}$ will be expressed as :

$$P_k(y_1, ..., y_M) = \begin{bmatrix} (\sum_{i=1}^{M} w_{i,1,k} y_i) + (\sum_{i=1}^{T} a_{i,1,k} \tau_{i,k}) + c_{1,k} \\ \vdots \\ (\sum_{i=1}^{M} w_{i,N,k} y_i) + (\sum_{i=1}^{T} a_{i,N,k} \tau_{i,k}) + c_{N,k} \end{bmatrix}, \begin{cases} min(L_{1,k}) \leqslant \tau_{1,k} \leqslant max(G_{1,k}) \\ \vdots \\ min(L_{T,k}) \leqslant \tau_{T,k} \leqslant max(G_{T,k}) \\ min(L'_{1,k}) \leqslant \tau'_{1,k} \leqslant max(G'_{1,k}) \\ \vdots \\ min(L'_{T',k}) \leqslant \tau'_{T',k} \leqslant max(G'_{T',k}) \end{cases}$$

Which can be simplify as :

$$P_k(\mathbf{y}) = \mathbf{W}_k \mathbf{y} + \mathbf{A}_k \boldsymbol{\tau}_k + \mathbf{c}_k, \begin{cases} \mathbf{L}_k \leqslant \boldsymbol{\tau}_k \leqslant \mathbf{G}_k \\ \mathbf{L'}_k \leqslant \boldsymbol{\tau'}_k \leqslant \mathbf{G'}_k \end{cases}$$

3.4 Extension to simple ReLU activation

ReLU preimage can be described as follow :

As a reminder :

$$ReLU : \mathbb{R} \longrightarrow \mathbb{R}^+$$

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Let $iRL : \mathbb{R} \longrightarrow \mathbb{R}^{\mathbb{R}}$, the preimage of ReLU.

$$iRL(y) = \begin{cases} y, & \text{if } y > 0 \\ ]-\infty, 0], & \text{otherwise} \end{cases}$$

Inverting a ReLU layer is possible thanks to the same method as previously seen. In addition to it, a negative real variable must be added for each inequality solved as less than zero for a given set.

4 Generalized Preimage

Neural networks can be described as ordered weights associated with activation function. Each layer's weights can be formalized as matrices. It means that all ANN weights can be expressed as a vector $n+1$ matrices, where n is the number of hidden layers.

Let $\{\mathbf{W}_i\}_{i \in [\![1,n+1]\!]}$, the family of weights matrice of a neural network.
Let $\mathbf{W} = \begin{bmatrix}\mathbf{W}_1 & \cdots & \mathbf{W}_{n+1}\end{bmatrix}^{\mathbf{T}}$, the vector of weight matrices.

The generalized preimage $P(\mathbf{y}, \mathbf{W})$ is an expression of the preimage of all neural networks that have the same activation functions and also the same dimension for the matrix $\mathbf{W}$ and it element.
Notice that P can be computed using the same way as seen previously. It can be done by considering each weight as a variable instead of a constant. The only information that has to be known is the sign of each weight. This is mandatory during the elimination of variables when solving inequalities system.

$\forall i \in [\![1, n+1]\!], \forall w \in \mathbf{W}_i, w' = sgn(w)|w|$, where sgn is the sign function.

If $sgn(w)$ is known then the preimage can be computed by considering $|w|$ as a positive variable. Note that the exact same function will result of this computation for neural network with same sign for each weight.
This result shows that neural networks with the same sign for each weight have similar properties such as the same generalized preimage.

5 Complexity

5.1 Space Complexity

The space complexity is directly linked to the number of solutions. In a piecewise linear case, the number of solutions has been given in section 3.3 which was equal to $\Omega = 2^{N+\sum_{k=1}^{H} N_k}$,where N is the length of the outputs layer, the N_k of the k-th hidden layer and H the total number of hidden layer. It gives a space complexity of $O(\Omega) = O(2^{N+\sum_{k=1}^{H} N_k})$.
The algorithm space complexity is exponential.

5.2 Time Complexity

The algorithm time complexity depends on the linear inequalities solver. Let f the time complexity function of the linear inequalities systems solver.
Let n, m respectively the number of inequalities and its number of variables. Then f is a function of n, m such that the time complexity is equal to $O(f(n, m))$.
For a given multilayer neural network, let S the sum of the length of the hidden layers and the output layer.
Let L, N respectively be the length of the first hidden layer of the output layer. Then the time complexity is equal to : $O(2^S f(L, L-N))$. In that case, 2^S is the total number of system to solve for the first hidden layer, and $L-N$ the number of free variables.

The algorithm depends on the Fourier-Motzkin time complexity. The Fourier-Motzkin time complexity is in $O(n^{2^m})$ in the worst case (Where n is number of inequalities and m the number of variables to eliminate) [7]. Because of the number of solution that growth exponentially, the time complexity of the algorithm is expressed as : $O(2^S L^{2^{L-N}})$, where S is the sum of the length of the hidden layer layers and the input layer, N the length of the output layer, L the length of the first hidden layer.
The total time complexity of the algorithm is double exponential.

6 Conclusion

Computing the preimage of ANN with linear activation for hidden layers can be reduced to a matrix problem. It ensures that solution sets can be found thanks to analytical calculation and Gaussian elimination. For the nonlinear case, the method can't invert networks with non-piecewise linear hidden layers activation like tanh or sigmoid in general cases. Notice that it is still possible in specific cases or if the non-piecewise linear activation concern the output layer.
With piecewise linear functions like ReLU or Parametric ReLU, it become possible to analytically describe the preimage of any kind of FFNN. This has been done thanks to the fact that preimages of piecewise linear functions are piecewise linear themselves and can be separated into linear subproblems if the sign of their inputs are known. These signs can be found by solving a system of linear inequalities thanks to Fourier-Motzkin elimination. Every analytical solver for multivariate linear inequalities system would have the exact same result.
Notice that even with more outputs than inputs, the method can find and return the preimage of the closest valid output. This can be done through least squares error minimization.

This study shows that ANN that can approximate any continuous function can be fully inverted analytically. Due to the exponential time complexity for the computation of the preimage because of the growing number of sets of solutions, it can seem that this approach is not usable in practice for large neural networks. However, for practical application, most of the time only a small part of all solutions will be necessary. It means that computation time may be reduced by not computing the entire set of solutions.

Exact preimage computation will give a better understanding of choices taken by FFNN. It also enable to better control the behavior of ANN that are often used as blackbox function.
As a resume , this study shows a method that setup a base for FFNN preimage computation thanks to a straight-forward algorithm for analytical solving. It ensures that exact preimages of FFNN can be theoretically found and open the way to other ANN inverting based studies.

References

[1] Stefan Carlsson, Hossein Azizpour, and Ali Razavian. “The preimage of rectifier network activities”. In: (2016).

[2] Ashima Dua and Amar Gupta. *Inversion of neural networks: A solution to the problems encountered by a steel corporation.* 2000.

[3] Boris Hanin and Mark Sellke. “Approximating continuous functions by relu nets of minimal width”. In: *arXiv preprint arXiv:1710.11278* (2017).

[4] Humayun Kabir et al. “Neural network inverse modeling and applications to microwave filter design”. In: *IEEE Transactions on Microwave Theory and Techniques* 56.4 (2008), pp. 867–879.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[6] Kyle Matoba and François Fleuret. “Computing Preimages of Deep Neural Networks with Applications to Safety”. In: (2020).

[7] Theodor Motzkin. *The theory of linear inequalities.* Tech. rep. RAND CORP SANTA MONICA CA, 1952.

[8] Terrence J Sejnowski. “The unreasonable effectiveness of deep learning in artificial intelligence”. In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30033–30038.

Appendices

A Fourier-Motzkin Elimination

Fourier-Motzkin elimination [7] is an algorithm which eliminate variables in multivariate linear inequalities systems.
In a general case, linear inequalities system can be written as:

$$\mathbf{W}\mathbf{x} \geqslant 0$$

Algorithm 1 Fourier-Motzkin algorithm

```
 Inputs : $\mathbf{W}$,$\mathbf{x}$
while $\exists(w_{i_1,j}, w_{i_2,j}), (i_1, i_2) \in [\![1, N]\!]^2, j \in [\![1, M]\!], w_{i_1,j}w_{i_2,j} < 0)$ do
  $plus \leftarrow []$; (Where [] is an empty list)
  $minus \leftarrow []$;
  $other \leftarrow []$;
  for $i \leftarrow 1$ to $N$ do
    if $w_{i,j} \neq 0$ then
      if $w_{i,j} > 0$ then
        $plus$ add $(\frac{1}{w_{i,j}})W_i$; (Where $W_i$ is the $i$-th lign of the matrix $\mathbf{W}$)
      else
        $minus$ add $(\frac{1}{w_{i,j}})W_i$;
      end if
    else
      $other$ add $W_i$;
    end if
  end for
  $N \leftarrow card(minus)card(plus) + card(other)$; ($card$ the cardinal function)
  $M \leftarrow M - 1$;
  $\mathbf{W} \leftarrow 0_{N,M}$
  $i \leftarrow 1$;
  for $eq_1$ in $plus$ do
    for $eq_2$ in $minus$ do
      $W_i \leftarrow eq_1 - eq_2$
      $i \leftarrow i + 1$;
    end for
  end for
  for $eq$ in $other$ do
    $W_i \leftarrow eq$
    $i \leftarrow i + 1$;
  end for
end while
return $\mathbf{W}$
```

Let $k \in [\![1, N]\!]$
Let $P, Q \in \mathbb{N}, P + Q = M$

With Fourier-Motzkin algorithm, each eliminated variable x_k would be expressed as follows:

$$S' = \begin{cases} x_k \geqslant -\frac{1}{w_{k,1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,1}x_i) + b_1] \\ x_k \geqslant -\frac{1}{w_{k,2}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,2}x_i) + b_2] \\ \vdots \\ x_k \geqslant -\frac{1}{w_{k,P}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P}x_i) + b_P] \\ x_k \leqslant -\frac{1}{w_{k,P+1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] \\ x_k \leqslant -\frac{1}{w_{k,P+2}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+2}x_i) + b_{P+2}] \\ \vdots \\ x_k \leqslant -\frac{1}{w_{k,P+Q}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] \end{cases} \tag{12}$$

$$\Rightarrow \begin{cases} x_k \geqslant max_{j\in[\![1,P]\!]}(-\frac{1}{w_{k,j}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,j}x_i) + b_j])) \\ x_k \leqslant min_{j\in[\![P+1,P+Q]\!]}(-\frac{1}{w_{k,j}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,j}x_i) + b_j])) \end{cases}$$

The next step is to keep (12) consistent. If consistent solutions exist, they will be able to be found by processing the same Fourier-Motzkin elimination process on the newly introduce inequalities system S'. According to (12), inequalities to satisfy can be rewritten as :

$$S' = \begin{cases} -\frac{1}{w_{k,P+1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] \geqslant -\frac{1}{w_{k,1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,1}x_i) + b_1] \\ \vdots \\ -\frac{1}{w_{k,P+Q}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] \geqslant -\frac{1}{w_{k,1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,1}x_i) + b_1] \\ -\frac{1}{w_{k,P+1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] \geqslant -\frac{1}{w_{k,2}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,2}x_i) + b_2] \\ \vdots \\ -\frac{1}{w_{k,P+Q}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] \geqslant -\frac{1}{w_{k,2}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,2}x_i) + b_2] \\ \vdots \\ \\ \vdots \\ -\frac{1}{w_{k,P+1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] \geqslant -\frac{1}{w_{k,P}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P}x_i) + b_P] \\ \vdots \\ -\frac{1}{w_{k,P+Q}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] \geqslant -\frac{1}{w_{k,P}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,P}x_i) + b_P] \end{cases}$$

$$\Rightarrow S' = \begin{cases} -\frac{1}{w_{k,P+1}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] + \frac{1}{w_{k,1}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,1}x_i) + b_1] \geqslant 0 \\ \vdots \\ -\frac{1}{w_{k,P+Q}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] + \frac{1}{w_{k,1}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,1}x_i) + b_1] \geqslant 0 \\ -\frac{1}{w_{k,P+1}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] + \frac{1}{w_{k,2}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,2}x_i) + b_2] \geqslant 0 \\ \vdots \\ -\frac{1}{w_{k,P+Q}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] + \frac{1}{w_{k,2}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,2}x_i) + b_2] \geqslant 0 \\ \vdots \\ \\ \vdots \\ -\frac{1}{w_{k,P+1}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P+1}x_i) + b_{P+1}] + \frac{1}{w_{k,P}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P}x_i) + b_P] \geqslant 0 \\ \vdots \\ -\frac{1}{w_{k,P+Q}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P+Q}x_i) + b_{P+Q}] + \frac{1}{w_{k,P}}[(\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w_{i,P}x_i) + b_P] \geqslant 0 \end{cases}$$

Let $\{w'_{i,j}\}_{i\in\llbracket 1,N\rrbracket\setminus\{k\},j\in\llbracket 1,PQ\rrbracket} \in \mathbb{R}^{N+PQ-1}, w'_{i,j} = \frac{w_{i,\lceil j/Q\rceil}}{w_{k,\lceil j/Q\rceil}} - \frac{w_{i,P+1+(j-1[Q])}}{w_{k,P+1+(j-1[Q])}}$
Let $\{b'_j\}_{j\in\llbracket 1,PQ\rrbracket} \in \mathbb{R}^{PQ}, b'_j = \frac{b_{\lceil j/Q\rceil}}{w_{k,\lceil j/Q\rceil}} - \frac{b_{P+1+(j-1[Q])}}{w_{k,P+1+(j-1[Q])}}$

Then :

$$S' = \begin{cases} (\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w'_{i,1}x_i) + b'_1 \geqslant 0 \\ (\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w'_{i,2}x_i) + b'_2 \geqslant 0 \\ \vdots \\ (\sum_{i\in\llbracket 1,N\rrbracket\setminus\{k\}} w'_{i,PQ}x_i) + b'_{PQ} \geqslant 0 \end{cases}$$

As S' has the same form as S, the same process can be applied to eliminate another free variable.

Extrema Approach
After Fourier-Motzkin elimination, the linear inequalities system S remains with same signs real coefficients such as :

$$\forall (w_{i,j_1}, w_{i,j_2}), i \in \llbracket 1,N\rrbracket, (j_1,j_2) \in \llbracket 1,M\rrbracket^2, w_{i,j_1}w_{i,j_2} > 0$$

As a reminder :

$\forall i \in \llbracket 1,N\rrbracket, \forall j \in \llbracket 1,M\rrbracket, w_{i,j}, b_i \in \mathbb{R}$

$$S = \begin{cases} (\sum_{i=1}^{N} w_{i,1}x_i) + b_1 \geqslant 0 \\ (\sum_{i=1}^{N} w_{i,2}x_i) + b_2 \geqslant 0 \\ \vdots \\ (\sum_{i=1}^{N} w_{i,M}x_i) + b_M \geqslant 0 \end{cases}$$

To find solutions of S a similar process will be applied to it. In fact, it is a particular case of Fourier-Motzkin elimination where there is no opposite sign to proceed as previously. To solve this particular case, first select a variable $x_k \in \mathbf{x}, k \in [\![1, N]\!]$.
While there still remain unsolved inequalities, process as follows:

$\forall j \in [\![1, M]\!], w_{k,j} > 0$, then :

$(\sum_{i=1}^{N} w_{i,j}x_i) + b_j \geqslant 0 \Rightarrow x_k \geqslant -\frac{1}{w_{k,j}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,j}x_i) + b_j]$

Otherwise if $w_{k,j} < 0$,

$(\sum_{i=1}^{N} w_{i,j}x_i) + b_j \geqslant 0 \Rightarrow x_k \leqslant -\frac{1}{w_{k,j}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,j}x_i) + b_j]$

The variable x_k can be eliminated from the system as it can be expressed as follows:

$$\begin{cases} x_k \geqslant -\frac{1}{w_{k,1}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,1}x_i) + b_1] \\ x_k \geqslant -\frac{1}{w_{k,2}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,2}x_i) + b_2] \\ \vdots \\ x_k \geqslant -\frac{1}{w_{k,n}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,n}x_i) + b_n] \end{cases} \tag{13}$$

$$\Rightarrow x_k \geqslant max_{j\in[\![1,M]\!]}(-\frac{1}{w_{k,j}}[(\sum_{i\in[\![1,N]\!]\setminus\{k\}} w_{i,j}x_i) + b_j]))$$

Notice that the case where $w_{k,j} < 0$ can be solved with the same method by reversing the sign of the previous inequalities and by replacing the *max* function by a *min* function.

At this point, each variable left is free or expressed by trivial inequalities that can be reduced as a maximum (or minimum) evaluation. For example, let v a remaining variable.

Let $n, m \in \mathbb{N}, \mathbf{c} \in \mathbb{R}^{n+m}$

$$\begin{cases} v \geqslant c_1 \\ v \geqslant c_2 \\ \vdots \\ v \geqslant c_n \\ v \leqslant c_{n+1} \\ \vdots \\ v \leqslant c_{n+m} \end{cases} \Rightarrow \begin{cases} v \geqslant max(c_i, i \in [\![1, n]\!]) \\ v \leqslant min(c_j, j \in [\![n+1, n+m]\!]) \end{cases} \tag{14}$$

As $\mathbf{c}$ is a vector of n known real values, evaluating v can be done in $O(n)$ time complexity. Notice that if inequalities of (14) are inconsistent, the system S has no solutions. Because every variables are expressed as system of inequalities of known symbol, solutions of S can be found by iteratively computing the maximum (or minimum) value for each of them.
The previous algorithm gives a convenient way to solve linear inequalities as it ends in an exponential time complexity. It however needs more computation (in polynomial time) to enable its solutions to be exploitable.

Please find the associated code : `https://github.com/TheoN70/preimage_nn`