Internship Report : Neuro-fuzzy Networks with Mixed Weights

Thomas Sesmat
Supervised by Jean-Loup Farges, Gauthier Picard, and Filipo Perotto

November 15, 2023

Acknowledgements

These acknowledgements are not ranked by importance.

I would like to express my sincere gratitude to Mr. Philippe Villedieu, Mrs. Béatrice Laurent Bonneau, as well as the internship service at INSA and the GMM department for their availability and, above all, their swift responsiveness at the beginning of this internship, which started off with considerable difficulty.

I am very warmly thankful to Mr. Jean-Loup Farges for his immense help at every moment, his exceptional teaching skills, and his patience with me.

I am also grateful to Mr. Filipo Perotto for his assistance and for caring about the well-being of his temporary office co-worker.

My thanks go to Gauthier Picard for his support and kindness.

I would like to thank the entire ONERA administration and the DTIS service for their warm welcome and my smooth integration.

Finally, I thank my parents for their constant availability and unwavering support.

Once again, I would like to thank my three mentors, Mr. Jean-Loup Farges, Mr. Filipo Perotto, and Mr. Gauthier Picard, for their help, kindness, and trust in me, which allowed me to feel truly empowered to contribute throughout this internship.

:)

Résumé

Increasingly, studies are raising alarms about the importance of interpretability in artificial intelligence. The potential for reasoning biases and discrimination restricts the application of machine learning in critical fields such as aviation or healthcare, where learned models must be transparent.

Despite the growing number of research groups focused on this issue, no study so far has managed to develop an inference algorithm that provides satisfactory transparency regarding its decision-making processes.

This internship focused on investigating a promising approach : fuzzy neural networks with mixed weights. These are inference algorithms that combine neural networks with fuzzy logic. The integration of fuzzy logic and the design of adaptable rules require the use of Boolean weights in certain layers of the network, while other parameters in this network are continuous variables. The training of this hybrid network is thus based on mixed-variable optimization.

This internship led to the design of a learning algorithm incorporating an adapted Tabu search along with classic backpropagation. Additionally, considerations are presented and explored regarding a method for translating the network's various weights after training. The results obtained at the end of this internship provide partial answers to many questions surrounding this issue and offer promising perspectives. This approach remains under active development to address the identified challenges to date.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Contexte

This report covers a Master's internship conducted at ONERA (French Aerospace Lab) from July to October 2023. In this report, I formally describe my work on the development of a fuzzy neural network with mixed weights. The environment, state of the art, and the topic of the internship are introduced in this chapter, followed by a description of the improvements made to the initial network in Chapter2. Next, the learning algorithm developed for this network is explained in Chapter 3. The various results obtained and the study of the influence of global parameters are detailed in Chapter 5. Finally, Chapter6 is dedicated to the conclusions on the subject and the internship.

1.1 Overview of the Environment

1.1.1 ONERA – *The French Aerospace Lab*

Founded in 1946, ONERA (Office National d'Etudes et de Recherches Aérospatiale) is the French aerospace research center. Its missions are as follows :

— To develop and lead research in the aerospace field ;
— To design, implement, and carry out the management of this research ;
— To ensure the dissemination of research results at national and international levels, in coordination with services or organizations responsible for scientific and technical research ;
— To promote the valorization of these results by the aerospace industry ;
— To facilitate their potential application outside the aerospace field.

ONERA has around 2,000 employees, including 1,500 researchers, engineers, and technicians spread across eight sites in France, as shown in Figure 1.1. To address the major challenges of the aerospace and defense industries, ONERA has equipped itself with unique experimental facilities in Europe. All major civil and military aerospace programs in France and Europe carry part of ONERA's DNA : Ariane, Airbus, Falcon, Rafale, missiles, helicopters, engines, radars, etc.

1.1.2 Scientific Department and Research Unit

Research conducted at ONERA is organized into four branches corresponding to major disciplinary fields : Fluid Mechanics and Energy, Physics, Materials and Structures, and Information Processing and Systems. Within each branch, several scientific departments operate, contributing to the diversity of research activities. At the ONERA center in Toulouse, we can highlight the following departments with their specific research domains :

— DEMR : Electro-Magnetism and Radar
— DMPE : Multi-Physics for Energy

FIGURE 1.1 – ONERA centers in France (2018)

— DOTA : Optics and Associated Techniques
— DPHY : Physics, Instrumentation, Environment, and Space
— DTIS : Information Processing and Systems

DTIS

The Information Processing and Systems Department (DTIS), where I completed my internship, conducts studies and research to master the design, operation, and autonomy of aerospace systems. The department applies its expertise in the fields of aeronautics, space, and defense, with the following primary applications :

— Aircraft (transport planes, fighter jets, drones, airships, *etc.*)
— Aerospace systems (air transport systems, launchers, satellites, *etc.*)
— Information systems (surveillance systems, localization systems, *etc.*)
— Defense systems (missiles, systems of systems, *etc.*)

SYD

Within DTIS, I was part of the Intelligent Systems and Decision Unit (SYD), which focuses on discrete formal methods for problem-solving, multi-agent systems, and requirements modeling. The primary areas studied within this unit include Operations Research, Artificial Intelligence (AI), requirements engineering, planning, algorithms and combinatorics, and collaborative decision-making and allocation. My supervisors during this 12-week internship were Jean-Loup FARGES, Filipo PEROTO, and Gauthier PICARD.

1.2 State of the Art

Since its beginnings in the 1950s, with Warren McCulloch and Walter Pitts' work on cybernetics and contributions from John Von Neumann and Alan Turing on binary logic computing, research in AI has traditionally been classified into two main approaches : symbolic AI and connectionist AI [1], [2].

Symbolic AI aims to formally reproduce human reasoning [3]. It uses fixed, primarily Boolean rules and symbols to define a logical sequence of reasoning. This approach was notably used in translation algorithms, where translating from one language to another is achieved through a set of

well-defined rules (although it has now been largely replaced in this area by connectionist AI). By introducing human-comprehensible expert rules (if "event," then "consequence"), symbolic AI offers high interpretability due to the transparency of the systems it produces, making its processes easy to understand. Widely used from the 1970s to the 1990s, it was eventually sidelined due to its rigidity in decision-making and the challenge of handling increasingly complex situations, where each rule must be defined by a human.

Connectionist AI, or Neural Networks (NNs), seeks to replicate human brain functioning. NNs have the capacity to "learn" the mathematical relationships between a series of input and corresponding output variables. These networks are trained to adjust their internal weights according to the mathematical relationships identified between inputs and outputs on a known dataset [4]. They gained popularity in the 1980s with the invention of the backpropagation algorithm [5]. Since then, numerous different networks have been implemented, and NNs are now widely used in fields such as image recognition, signal processing, and simulation. They enable high precision, but being more complex, NNs are often seen as "black boxes." Justifying and interpreting the configuration of a network beyond performance metrics, like accuracy gains, is challenging.

This lack of interpretability imposes concrete limitations on the use of NNs in critical fields such as aeronautics, medicine, and finance. Furthermore, with scandals related to AI biases (e.g., gender or racial biases stemming from training data or algorithm design) and the identification of significant biases during learning [6, 7], the current global trend is to develop programs implementing certain ethical rules. It is thus essential to be able to justify a program's functionality. Numerous research groups have been established in recent years to address these issues [8, 9, 10]. The AI Act, European-level legislation, has also been under development for several years to regulate the use of artificial intelligence [11].

1.2.1 Fuzzy Logic

Fuzzy logic is a set theory introduced by Lotfi A. Zadeh in 1965 [12], extending binary logic by accounting for degrees of truth. It enables nuanced representation of uncertain data and complex decision-making processes, making it particularly useful for translating human decision logic. Fuzzy logic is based on four fundamental elements : fuzzy sets, membership functions, fuzzy logic operations, and fuzzy inference.

Fuzzy Set

In classical set theory, an element either belongs to a set (with a membership value of 1) or does not belong (with a membership value of 0). However, for a support set X, a fuzzy set A is defined over a discourse universe $\mathbb{D}$ as the application of a function from X to the interval [0, 1]. Each element x in X is assigned a membership degree, denoted as $\mu_A(x)$, representing the extent to which x belongs to A. Formally, a fuzzy set A can be represented as follows :

$$A = \{(x, \mu_A(x)) | x \in X\}$$

Membership Function

A membership function $\mu_A(x)$ quantifies the degree to which an element x belongs to a fuzzy set A. It characterizes how strongly x belongs to the set in a gradual way. When X corresponds to a set of real-valued variable vectors, membership functions can adopt different mathematical forms, including triangular, Gaussian, or sigmoid. The choice of these functions depends on the nature of the problem being modeled and the desired representation of uncertainty.

	Gödel T-norm	Product T-norm
Fuzzy AND	$\mu_C(x) = \min(\mu_A(x), \mu_B(x))$	$\mu_C(x) = \mu_A(x) \cdot \mu_B(x)$
Fuzzy OR	$\mu_C(x) = \max(\mu_A(x), \mu_B(x))$	$\mu_C(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$
Fuzzy NOT	$\mu_C(x) = 1 - \mu_A(x)$	

TABLE 1.1 – Logical operators for the Gödel T-norm and the Product T-norm. "$\cdot$" denotes the real numerical product.

Fuzzy Logic Operations

Fuzzy logic operations extend Boolean operations to adapt them to membership degrees. There are several definitions for logical operators based on the triangular norm (T-norm) used. Table 1.1 shows the expressions for the operators for the two most commonly used T-norms : the Gödel T-norm (or minimum norm) and the product T-norm :

— Fuzzy AND ($\wedge$) : For two fuzzy sets A and B with membership functions $\mu_A(x)$ and $\mu_B(x)$, the **Fuzzy AND** operation produces a new fuzzy set C with the membership function

$$\mu_C(x) = T(\mu_A(x), \mu_B(x))$$

where T is the T-norm considered.

— Fuzzy OR ($\vee$) : For fuzzy sets A and B, the **Fuzzy OR** operation produces a new fuzzy set C with the membership function $\mu_C(x)$ defined as follows :

$$\mu_C = T * (\mu_A(x), \mu_B(x))$$

where $T*$ is the T-conorm associated with T.

— Fuzzy NOT ($\neg$) : For a fuzzy set A with a membership function $\mu_A(x)$, the **Fuzzy NOT** operation negates it to create a new fuzzy set C with the membership function $\mu_C(x)$ defined as follows :

$$\mu_C(x) = 1 - \mu_A(x)$$

This is the canonical negation operator, with an expression common to all T-norms.

The product T-norm is differentiable and based on the assimilation of membership to a frequency, the representativity of frequencies, and the independence of memberships. The Gödel T-norm is the only T-norm exhibiting properties of idempotence, absorption, and distributivity [13]. However, it is not differentiable when $\mu_A(x) = \mu_B(x)$.

Fuzzy Inference

Fuzzy inference is a process that uses fuzzy logic rules and input data to deduce a crisp output. It involves the following steps :

— Fuzzification : Converts crisp input data into fuzzy sets by applying suitable membership functions.

— Rule Evaluation : Applies fuzzy logic rules that relate input fuzzy sets to output fuzzy sets through fuzzy logic operators.

— Aggregation : Combines output fuzzy sets to produce a composite fuzzy set that represents the full set of established rules for all possible outputs.

— Defuzzification : Converts the composite fuzzy set into a crisp value. The goal is to transform the previous layers' results into a numeric output. According to JSR. Jang, there are three types [14] : the weighted average of the net output for each rule derived from previous layers, the maximum of the previous layers' net outputs, or using Takagi-Sugeno *if-then* rules [15].

1.2.2 Backpropagation and Gradient Descent

Backpropagation is a method for efficiently calculating gradients for each layer composed of numerical variables in a neural network. It works recursively, propagating the network's error from output to input. Algorithm 1 presents in pseudocode the key steps in gradient backpropagation, integrating gradient descent.

Algorithme 1 : Gradient Backpropagation

Input : data - network training data, corresponding to input scores and their respective output vectors
Input : step - learning rate for gradient descent
Input : var - vector of the network's numerical variables
Output : trained_var - trained numerical variables of the network
grad_cum ← 0;
for *(x, y) ∈ data* **do**
    y_pred ← ForwardPropagation(x);
    grad_cum ← BackwardPropagation(y_pred, y);
end
grad ← grad_cum/data.size;
trained_var ← GradDescent(var, step, grad);

In the first step, for each pair of input-output vectors, the network uses the *ForwardPropagation()* function to transform the input into a predicted output through the network's layers. Once the prediction is obtained, *BackwardPropagation()* calculates the gradients of the error relative to each numerical parameter (i.e., numerical weights and biases) of the model using the chain rule. The resulting value is stored in *grad_ cum*. After all the data has been processed, the mean gradient *grad* is obtained by dividing *grad_ cum* by the number of data pairs. Finally, the *GradDescent()* function performs gradient descent to adjust the model's numerical parameters, defined as :

$$trained\_var = var - step * grad$$

When training a neural network, Algorithm 1 is repeated for a set number of iterations (epochs) or until a stopping criterion is reached, such as when the cost function meets an acceptable threshold or when the difference between successive iterations is below a certain value.

1.2.3 Tabu Method

Here we present a technique that is not used exactly as described in our network but from which we draw inspiration. The Tabu algorithm is an optimization and heuristic search method aimed at finding the best possible solution in a complex search space while avoiding getting stuck in local optima. A description of the key steps in the method is presented in Algorithm 2.

Starting from a solution s_best considered ideal in its current state, $GenerateNeighbor()$ performs a local search around the current solution to identify neighboring solutions $neighbor\_s$. Among all neighbors, $SelectBestNonTabu()$ selects the best non-tabu solution, i.e., one that has not been recently explored according to the *antiquity* criterion. Then, $UpdateTabu$ updates the *tabu* list by adding the recently explored solution and removing those that are no longer tabu according to the same *antiquity* criterion. The current ideal solution is replaced by its best neighbor if it leads to a better result. Otherwise, we keep this solution and search for a new neighbor with the updated *tabu* list. At the end of the loop, the stopping criterion is updated by the $UpdateCrit()$ function.

Algorithme 2 : Tabu Method in Optimization

Input : s_ini - the initial solution
Input : antiquity - antiquity criterion for the Tabu list
Input : stop_crit - stopping criterion (e.g., number of iterations, minimum difference between two considered solutions)
Output : s_best - the best solution found once the stopping criterion is reached

```
s_best ← s_ini;
tabu ← None;
while not stop_crit do
    neighbor_s ← GenerateNeighbor(s_best);
    best_neighbor ← SelectBestNonTabu(neighbor_s);
    tabu ← UpdateTabu(tabu, best_neighbor, antiquity);
    if best_neighbor < s_best then
        s_best ← best_neighbor;
    end
    stop_crit ← UpdateCrit(stop_crit);
end
```

1.2.4 Neuro-Fuzzy Networks (NFN)

Several avenues have been explored to advance XAI (Explainable AI) [16], [17]. A possible method is neuro-fuzzy networks [14]. These are hybrid networks situated between symbolic AI and connectionist AI, incorporating Zadeh's fuzzy logic into neural networks. Integrating this logic into neural networks allows us to benefit from both advantages : handling imprecision and modeling reasoning similar to the human brain, making it more interpretable. NFNs manipulate linguistic descriptors rather than numerical inputs, with fuzzy rules describing relationships between these descriptors. The network learns from examples and error correction, as in traditional neural networks, but uses fuzzy membership functions to describe inputs. Their structure is inspired by fuzzy inference schemes, with successive layers of the network performing fuzzification, inference, and defuzzification as previously described. A recent review of the literature on this topic concludes that neuro-fuzzy network models and their derivatives effectively build systems with a high degree of accuracy that can be interpreted at an appropriate level across many fields in economics and sciences [18].

However, very few studies so far focus on the dynamic selection of network learning rules to improve the initial expert rule. Notable is the study by G. Marra et al. [19], which presents a DCR (Deep Concept Reasoner) network capable of selecting rules based on an embedding and yielding excellent results in both rule interpretation and classification performance. However, using embeddings introduces two issues : a non-trivial amount of work is required on data (of any type) before using the DCR network, and the resulting outputs are not necessarily interpretable by humans due to the embedding. Thus, there is a risk that the problem is merely "shifted" rather than resolved.

The objective of this internship is to contribute to a project aimed at improving and learning rules by studying the concept of dynamic improvement of expert rules on a simple network through the optimization of Boolean weights. These weights configure this rule in addition to the numerical weights and biases. We also aim to make minimal transformations on the data prior to the network, in order to maintain interpretability and minimize overall computational time. The initial project aims to address the following scientific challenges :

— Precisely defining functions for certain nodes in the network and characterizing the derivatives required for backpropagation with these functions.

— Treating the learning problem as a mixed-variable problem.
— Defining simplification rules to produce coherent and interpretable rules for humans.

With this goal in mind, we also aim to fully control the implementation by minimizing the use of prebuilt function libraries (such as *Scikit Learn*, *Pytorch*, etc.), as much as possible.

1.2.5 Project Status at the Start of the Internship

This internship follows a long-term project at ENSEIHT (equivalent to a 4th-year research project at INSA). During this project, students had already coded part of the network structure. In the following, we will describe the project's status at the start of this internship. Despite the numerous observations in this section, I do not in any way intend to diminish the work of the students who conducted this long project ; instead, I simply aim to provide an objective overview of the project's state when I took it over for the internship to clearly present the various advances made during this time. I would also like to note that their initial code, as well as their report, enabled me to be quickly effective in this internship.

Mixed-Weight Network

The initial structure of the network for the *feedforward* part was implemented and is shown in Figure 1.3. It is a multilayer perceptron containing four hidden layers : Fuzzification, Conjunction, Disjunction, and Normalization. For each score, two descriptors are associated : one representing the case where "the score is high" and the other "the score is low," with sigmoid membership functions. A visualization of two types of membership functions is shown in Figure 1.2. The triangular norm used is Gödel's.

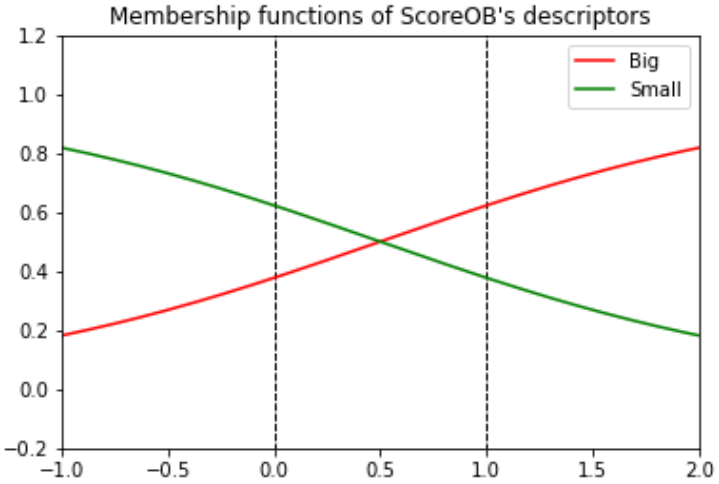


FIGURE 1.2 – Example : visualization of membership functions for the *orange box score* (airplane crash dataset)

Fuzzification Layer :

This first layer integrates fuzzy logic into the network, associating linguistic descriptors with various scores. Each linguistic descriptor is characterized by a sigmoid function of the form :

$$s(x) = \frac{1}{1 + e^{-(wx+b)}} \tag{1.1}$$

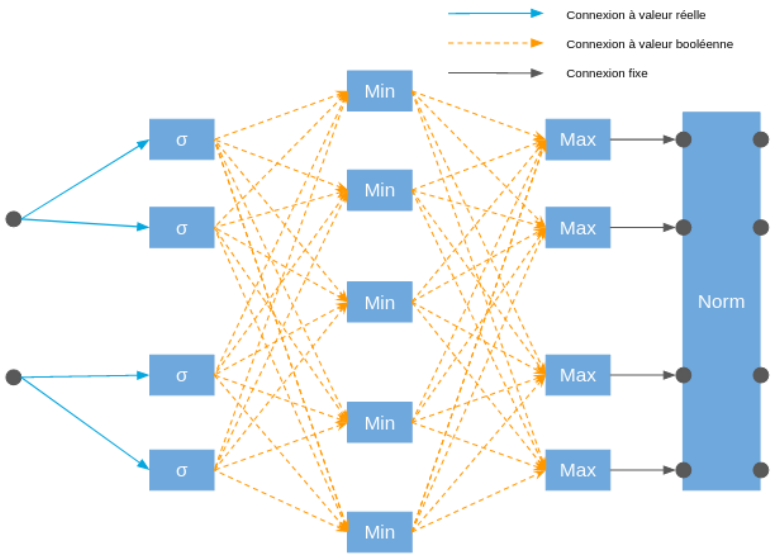


FIGURE 1.3 – Typical structure of a neuro-fuzzy network as implemented

where w and b are the weight and bias associated with this sigmoid. In our case, we consider two linguistic descriptors for each score, corresponding to *the score is high* and *the score is low*.

We decompose the layer as follows : it consists of blocks of sigmoids, with the number of blocks corresponding to the number of input scores, denoted here as n_{in}. Each block i consists of $n_{desc,i}$ sigmoids, corresponding to the number of descriptors associated with this block. To summarize, the fuzzification layer will return $n_{out}^{fuz} = \sum_{i=1}^{n_{in}} n_i^{desc}$ values where the j^{th} value of block i will be :

$$x_{i,j} = \frac{1}{1 + e^{-(w_{i,j}x_i + b_{i,j})}} \tag{1.2}$$

The algorithmic implementation is done as a class. We implement three nested classes :

— *SingleInputSigmoid*, initialized with a weight and a bias, creates a cell containing the information (weight and bias) of the considered descriptor for a given score. This class can calculate, through the *activate*() method, the membership value of a score relative to this descriptor.

— *SingleInputMultiSigmoidBlock*, initialized with a list of weights and biases, creates a list of cells (a block) of *SingleInputSigmoid* type. This class can calculate, through the *activate*() method that calls *SingleInputSigmoid*, the membership values of a given score for the associated descriptors.

— *FuzzificationLayer*, initialized with a list of lists of weights and biases, creates the entire fuzzification layer via a list of blocks of *SingleInputMultiSigmoidBlock* type. This class can calculate, via the *activate*() method that calls *SingleInputMultiSigmoidBlock*, each score's set of membership values for the associated descriptors.

Conjunction Layer

This second layer combines the membership values obtained from the first layer using a pseudo-minimum on $[0; 1]$ defined as :

$$pseudo\_min(x, w^{conj}) = 1 + \min_j \left((x_j - 1)w_j^{conj}\right).$$

where w_j^{bool} is a list of Boolean weights that determine whether to include or exclude the different values x_j contained in vector x for the minimum calculation. This pseudo-minimum corresponds to the fuzzy logical **AND** value for Zadeh's fuzzy logic using Gödel's T-norm.

We decompose the layer as follows : it consists of logical conjunction cells, with the number of cells defined arbitrarily, denoted here as n_{conj}. Each weight associated with a cell corresponds to a vector of n_{out}^{fuzz} Boolean values. To summarize, the conjunction layer will return $n_{out}^{conj} = n_{conj}$ numerical values where the output of cell i will be :

$$x_{out,i}^{conj} = pseudo\_min(x_{out}^{fuzz}, w_i^{conj}) = 1 + \min_j \left((x_{out}^{fuzz} - 1) w_{i,j}^{conj} \right). \tag{1.3}$$

We implement two nested classes to define this layer :

— *AndCell*, initialized with a list of Boolean weights, creates a cell containing this information (weights). This class can calculate, through the *activate*() method, the pseudo-minimum value as defined in (1.3) based on the output of the previous layer.

— *AndLayer*, initialized with a list of Boolean weight lists, creates a list of *AndCell* cells. It represents the entire conjunction layer. This class can calculate, through the *activate*() method that calls *AndCell*, the pseudo-minimum values for all cells in the layer.

Disjunction Layer

This third layer combines the values from the previous layer using a pseudo-maximum on $[0; 1]$ defined similarly to the pseudo-minimum :

$$pseudo\_max(x, w^{disj}) = \max_j \left(((x_j) w_j^{disj}) \right).$$

where w_j^{disj} is a list of Boolean weights that determine whether to include or exclude the different values x_j contained in vector x for the maximum calculation. This pseudo-maximum corresponds to the fuzzy logical **OR** value for Zadeh's fuzzy logic.

We decompose the layer as follows : it consists of logical disjunction cells, with the number of cells equal to the number of classes in our problem, denoted here as n_{disj}. Indeed, in a more developed perceptron, the number of cells could vary, but here, only the normalization layer follows. Since the normalization layer does not change the number of outputs, it is necessary to have the correct number of outputs at this stage. Each weight associated with a cell corresponds to a vector of n_{out}^{conj} Boolean values. To summarize, the disjunction layer will return $n_{out}^{disj} = n_{disj}$ numerical values where the output of cell i will be :

$$x_{out,i}^{disj} = pseudo\_max(x_{out}^{conj}, w_i^{disj}) = \max_j \left(((x_{out}^{conj}) w_{i,j}^{disj}) \right). \tag{1.4}$$

We implement it in the same manner as the conjunction layer, using the maximum function instead of minimum.

Normalization Layer

This final layer normalizes the results from the previous layer to obtain, for each initially considered score vector of dimension n_{in}, a probability of belonging to each class. The normalization layer returns a vector of size n_{out}^{norm} equal to the number of classes in our problem. The i^{th} value of the output vector is :

$$x_{out,i}^{norm} = \frac{x_{out,i}^{disj}}{\sum_{j=1}^{n} x_{out,j}^{disj}}$$

where x_j is the j^{th} scalar associated with $x = (x_1, ..., x_n)$

We implement this layer as a single class. The *NormalizationLayer* class is initialized with the number of classes in our problem and can calculate, through the *activate*() method, the normalized value for each output from the previous layer.

Analytical Sensitivity Calculation

To improve our network, we will use backpropagation. Backpropagation involves propagating the network's output error through the various layers and adjusting the numerical variables accordingly. To accurately obtain the derivative of the backpropagation function, we will define the sensitivities of each layer's output with respect to its input. The chain rule then allows us to calculate each neuron's contribution to the error.

The error chosen is cross-entropy, denoted by CE and defined as follows :

$$CE = \frac{1}{n}\sum_{i=1}^{n} -y_i \log\left(x_i\right) \tag{1.5}$$

where y_i is the desired prediction, and x_i is the result obtained after forward propagation.

This function is well-suited to represent the error between two probability distributions, as its minimization corresponds to the logarithm of the maximum likelihood of the observations y_i for the distribution modeled by the network. For other problems, a generally used criterion is the mean squared error.

To obtain a value representing the entire sample, we average CE, our cost function, over all the values obtained for the dataset :

$$CE = \frac{1}{n^{data}}\sum_{k=1}^{n^{data}} H(x^{expc}(k), x^{norm}(k)) \tag{1.6}$$

where $x^{expc}(k)$ is the one-hot vector corresponding to the actual output for the k^{th} input vector.

For simplicity of notation, we consider only one sample in the remainder of this section.

Criterion Sensitivity

The sensitivity of the criterion with respect to x_i^{norm}, a parameter of the previous layer (normalization), is given by :

$$\frac{\partial CE(x^{expc}, x^{norm})}{\partial x_i^{norm}} = -\sum_{k=1}^{n^{norm}} \frac{\partial\left(x_k^{expc}\log(x_k^{norm})\right)}{\partial x_i^{norm}} \tag{1.7}$$

$$= -\frac{\partial\left(x_i^{expc}\log(x_i^{norm})\right)}{\partial x_i^{norm}} \tag{1.8}$$

$$= -\frac{x_i^{expc}}{x_i^{norm}} \tag{1.9}$$

By using the composition derivative formula, we then obtain :

$$\frac{\partial CE(x^{expc}, x^{norm})}{\partial z} = \sum_{i=1}^{n^{norm}} \frac{\partial CE(x^{expc}, x^{norm})}{\partial x_i^{norm}}\frac{\partial x_i^{norm}}{\partial z} \tag{1.10}$$

$$= -\sum_{i=1}^{n^{norm}} \frac{x_i^{expc}}{x_i^{norm}}\frac{\partial x_i^{norm}}{\partial z} \tag{1.11}$$

where z is the network parameter with respect to which we wish to derive CE.

Normalization Layer

We decompose the following derivative :

$$\frac{\partial x_i^{norm}}{\partial z} = \frac{\partial x_i^{norm}}{\partial x_j^{disj}} \frac{\partial x_j^{disj}}{\partial z} \tag{1.12}$$

with $(i, j) \in [\![1; n^{norm}]\!] \times [\![1; n^{disj}]\!]$.
NOTE : In practice, we have $n^{norm} = n^{disj}$ (see 1.3).

We derive the expression for the normalization layer :

$$\frac{\partial x_i^{norm}}{\partial x_j^{disj}} = \begin{cases} \frac{\sum_{k=1}^{n^{norm}} x_k^{disj}}{(\sum_{k=1}^{n^{norm}} x_k^{disj})^2} & \text{if } i \neq j; \\ \frac{x_i^{disj}}{(\sum_{k=1}^{n^{norm}} x_k^{disj})^2} & \text{otherwise.} \end{cases} \tag{1.13}$$

Disjunction Layer
Using Gödel's T-norm, the disjunction layer corresponds to the use of a pseudo-maximum, as defined in 1.2.1. Gradient backpropagation through this layer consists of propagating the gradient only at the index where the maximum value was obtained during forward propagation, setting the gradient of the other indices to zero. This propagates the gradient only to the single input that contributed to the maximum output, leaving other inputs unaffected.

Thus, for $(j, i) \in [\![1; n^{disj}]\!] \times [\![1; n^{conj}]\!]$:

$$\frac{\partial x_j^{disj}}{\partial z} = \frac{\partial x_j^{disj}}{\partial x_i^{conj}} \frac{\partial x_i^{conj}}{\partial z} \tag{1.14}$$

where

$$\frac{\partial x_j^{disj}}{\partial x_i^{conj}} = \begin{cases} 1 & \text{if } x_i^{conj} * w_{i,j}^{disj} \text{ is the maximum value} \\ 0 & \text{otherwise} \end{cases} \tag{1.15}$$

Conjunction Layer
Since the conjunction layer corresponds to a pseudo-min (see again 1.2.1), we find a result similar to the previous layer, for $(i, j, k) \in [\![1; n^{disj}]\!] \times [\![1; n^{conj}]\!] \times [\![1; n_j^{fuzz}]\!]$:

$$\frac{\partial x_i^{conj}}{\partial z} = \frac{\partial x_i^{conj}}{\partial x_{j,k}^{fuzz}} \frac{\partial x_{j,k}^{fuzz}}{\partial z} \tag{1.16}$$

$$\frac{\partial x_i^{conj}}{\partial x_{j,k}^{fuzz}} = \begin{cases} 1 & \text{if } x_{j,k}^{fuzz} \text{ is the minimum output value} \\ 0 & \text{otherwise} \end{cases} \tag{1.17}$$

Fuzzification Layer
We have two cases, depending on whether we derive with respect to weights or biases. In both cases, the result is a diagonal matrix with diagonal terms that are, for $(j, k) \in [\![1; n^{conj}]\!] \times [\![1; n_j^{fuzz}]\!]$:

$$\frac{\partial x_{j,k}^{fuzz}}{\partial w_{j,k}} = \frac{x_j}{1 + e^{-(w_{j,k}x_j + b_{j,k})}} \left(1 - \frac{1}{1 + e^{-(w_{j,k}x_j + b_{j,k})}}\right) \tag{1.18}$$

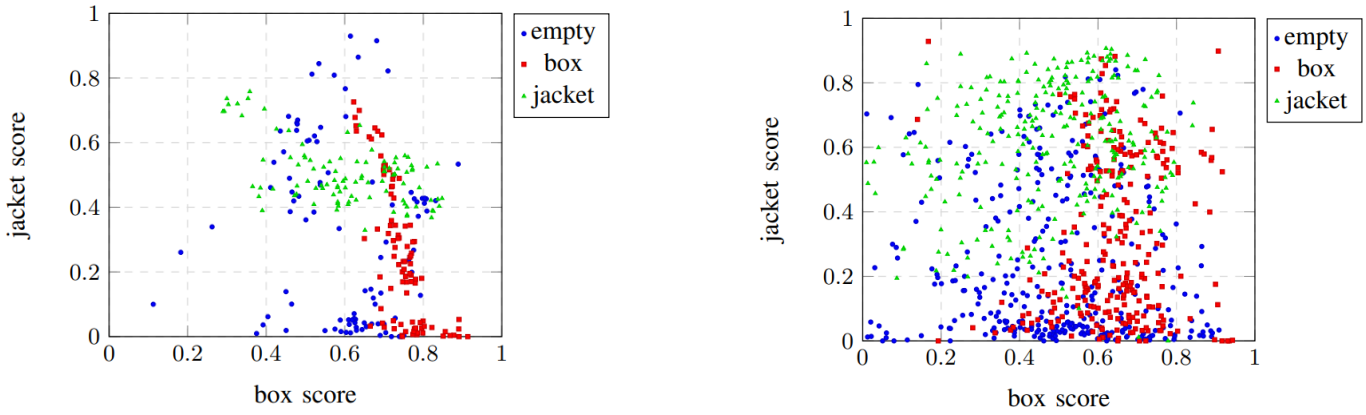
$$\frac{\partial x_{j,k}^{fuzz}}{\partial b_{j,k}} = \frac{1}{1 + e^{-(w_{j,k}x_j + b_{j,k})}} \left(1 - \frac{1}{1 + e^{-(w_{j,k}x_j + b_{j,k})}}\right) \tag{1.19}$$

Use Case

During the long-term project, all code was implemented in a Jupyter Notebook, and some codes exhibited both conceptual and structural errors (poor code optimization). The sensitivity calculation implementation was completed, and numerical sensitivity was also implemented. However, the sensitivity calculations were implemented outside of class structures, which did not fully leverage the benefits of object-oriented programming. Finally, both the analytical and numerical gradient calculations yielded differing results, indicating errors in the code, so they were not retained for future use.

The Data

To build this network, we initially based our work on a classification problem using a dataset modeling an airplane crash [20]. This dataset includes three classes : "Orange Box," "Yellow Jacket," and "Empty," with two scores to describe them : "Yellow Jacket Score" and "Orange Box Score." These scores were acquired by a drone that took photos of different areas, which then went through a series of instructions to transform them into scores. A detailed explanation of the entire acquisition process and data transformation to scores can be found in the original article [20]. The validation and training datasets were obtained from acquisitions taken at two different times, separated by several weeks.



(a) Training Data

(b) Validation Data

FIGURE 1.4 – Representation of classes based on scores.

Figure 1.4 shows the different classes based on the two scores for the validation and training data. There are 882 entries in the validation data, with each class approximately in a 1/3 proportion, and there are 287 entries in the training data, each class also in a 1/3 proportion. The data is split across six different files, containing only the score pairs according to the sample type (validation/training) and the class in question ("Orange Box," "Yellow Jacket," "Empty").

1.3 Conclusion

In this first part, we highlighted a gap in the scientific literature. Although other research is underway to address the issue, the results obtained do not convincingly solve the problem. Furthermore, the project's technical and mathematical concepts were clearly explained. Lastly, the state of the project at the start of the internship was outlined to distinguish progress made during this period from work previously completed by N7 students.

Chapitre 2

Improvements Made to the Network and Use Case

2.1 Data Modification

Since the initial dataset was acquired at two different times, we performed a Student's t-test with unequal variances to verify whether there were significant differences in the means between the two samples. The means and standard deviations for each score in the dataset are available in Appendix A.1 for the two samples considered. The p-values from these tests, shown in Table 2.1a, indicate a significant difference between the two datasets.

Classes	Score	p-value P
Box	Score OB	$P < 0.0001$
Box	Score YJ	$P = 0.0348$
Empty	Score OB	$P < 0.0001$
Empty	Score YJ	$P = 0.0055$
Jacket	Score OB	$P < 0.0001$
Jacket	Score YJ	$P < 0.0001$

(a) Initial Samples

Classes	Score	p-value P
Box	Score OB	$P = 0.9593$
Box	Score YJ	$P = 0.8072$
Empty	Score OB	$P = 0.7976$
Empty	Score YJ	$P = 0.5788$
Jacket	Score OB	$P = 0.3947$
Jacket	Score YJ	$P = 0.6062$

(b) Stratified Sampling

TABLE 2.1 – Student's t-test between training and validation samples for the airplane crash data

Thus, we merged and stratified the data to obtain more homogeneous samples (7/3 ratio for training/validation samples). A second Student's t-test, see Table 2.1b (means and variances also available in Appendix A.2), shows no significant differences between the sample means, and given the adopted method, we can conclude that these two samples are valid for network development.

The data format was also modified to have all classes associated with the same sample in a single file. The adopted format for an entry is as follows :

$$\text{Score OB} \mid \text{Score YJ} \mid \mathbb{1}_{OB} \mid \mathbb{1}_{YJ} \mid \mathbb{1}_{EB}$$

where OB corresponds to orange box, YJ to yellow jacket, and EB to empty.

After several weeks working with this dataset, we chose to focus on a more accessible dataset with less interwoven classes (see Figure 1.4). The selected dataset is "Breast Cancer Wisconsin (Diagnostic)." The data acquisition method is detailed in the original article [21]. Only certain variables were retained. The version used in this internship contains 7 variables and 559 instances. The different variables are :

— Diagnosis : {B : Benign, M : Malignant}, Output - Type of cancer detected. These are the two classes of interest.
— Radius (*Radius_ m*), Input - average distance from the center to perimeter points
— Texture (*Texture_ m*), Input - standard deviation of grayscale values
— Perimeter (*Perimeter_ m*), Input - tumor perimeter
— Area (*Area_ m*), Input - tumor area
— Concavity (*Concavity_ m*), Input - severity of concave parts of the contour

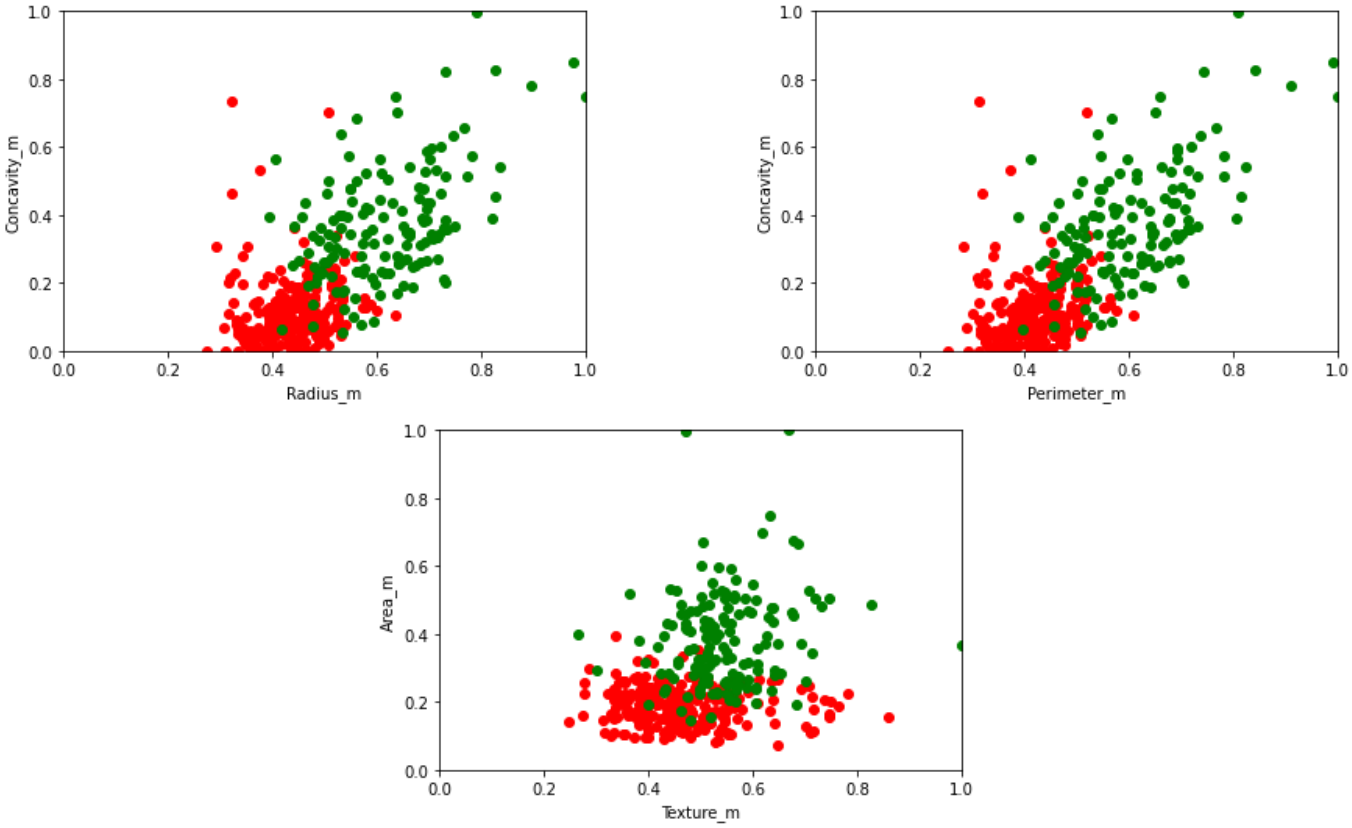


FIGURE 2.1 – Class representation based on scores for the Breast Cancer Wisconsin (Diagnostic) dataset. *Red* - Class B ; *Green* - Class M

The numerical variables were normalized to obtain a score. The data was formatted similarly to the previous dataset. Stratified sampling was also conducted to separate the data into a validation sample and a training sample (in the same proportions as the previous dataset). Since this dataset is relatively small, a Student's t-test was also conducted after sampling to ensure no significant differences between the training and validation samples. The means and standard deviations are available in Appendix A.3. Results can be observed in Table 2.2, allowing us to conclude that the stratified sampling produced suitable samples for training.

The visualization of classes based on input scores is also available in Figure 2.1. We observe that the classes can be relatively well separated for each score. Similar distributions are also observed for certain variables. The correlation plots in Figure 2.2 show that the radius is highly correlated with both perimeter and area. Since tumors are generally round, we see a linear correlation between perimeter and radius, as well as a quadratic correlation between radius and area.

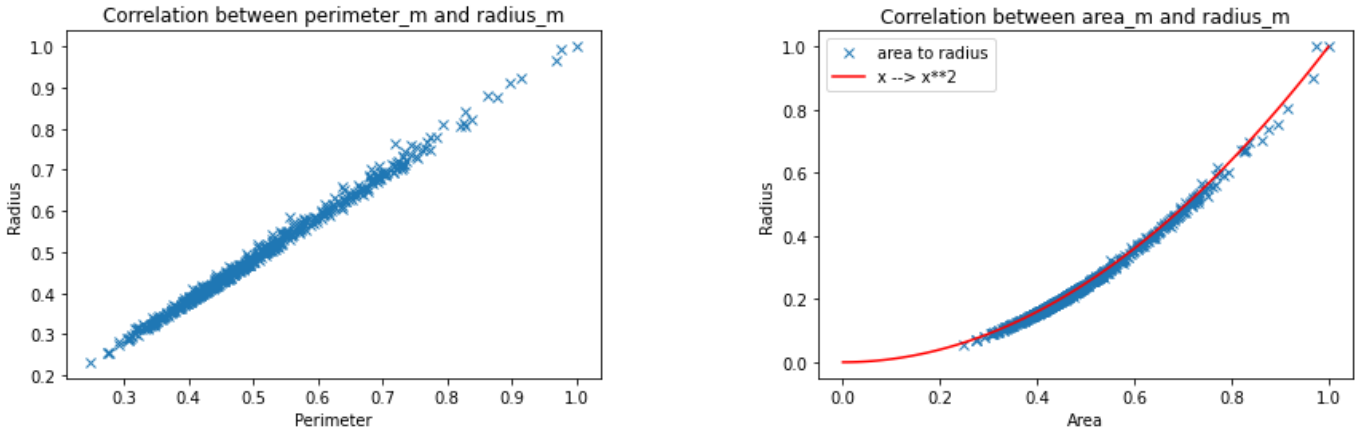


FIGURE 2.2 – Visualization of correlation between different scores in the Breast Cancer Wisconsin dataset

Class	Score	p-value P
B	Radius_m	P = 0.9826
B	Texture_m	P = 0.8888
B	Perimeter_m	P = 0.2288
B	Area_m	P = 0.6295
B	Concavity_m	P = 0.8579
M	Radius_m	P = 0.4038
M	Texture_m	P = 0.3942
M	Perimeter_m	P = 0.3478
M	Area_m	P = 0.5417
M	Concavity_m	P = 0.4473

TABLE 2.2 – Student's t-test between training and validation samples for the Breast Cancer Wisconsin dataset

2.2 Theoretical Modification

A verification of sensitivity calculations was conducted to ensure that the discrepancy between the numerical and analytical gradients developed during the Long Project was not due to a calculation error. Following this, the pseudo-maximum and pseudo-minimum formulas were slightly revised in their expressions to better reflect algorithmic reality :

Pseudo-minimum

$$pseudo\_min(x, w^{bool}) = \min_j \left(\left\{ x_j | w_j^{bool} = \top \right\} \cup \{1\} \right). \tag{2.1}$$

Pseudo-maximum

$$pseudo\_max(x_{out}^{conj}, w_i^{bool}) = \max_j \left(\left\{ x_j | w_{i,j}^{bool} = \top \right\} \cup \{0\} \right). \tag{2.2}$$

2.3 Overview of Algorithmic Structure

2.3.1 Feedforward

All network layers are interconnected and initialized within the *NeuroFuzzyNetwork* class, where the *activate*() method allows cascading calls to each layer's *activate*() methods to return the class membership probability for a given vector of scores.

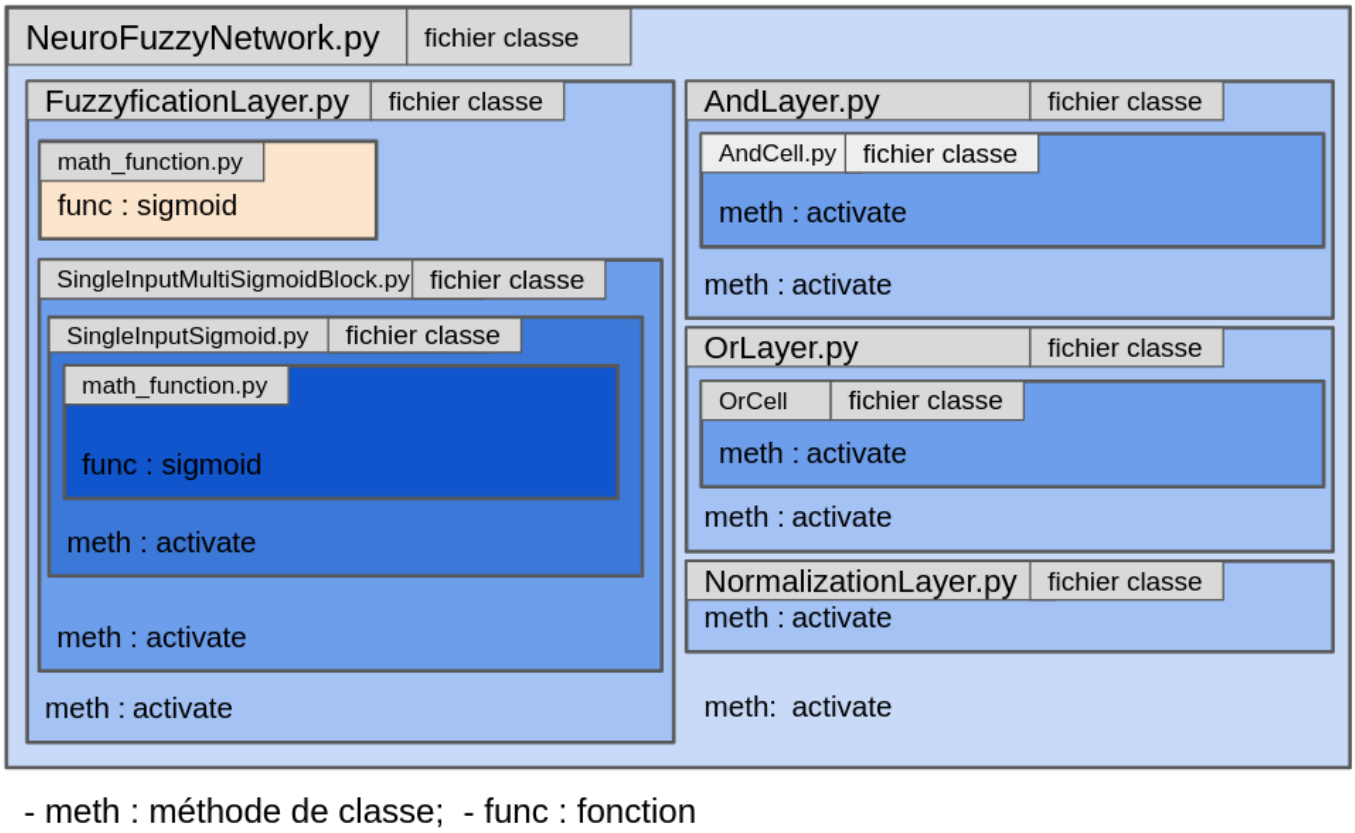


FIGURE 2.3 – Class Relationship Diagram for the Network Component

Figure 2.3 summarizes the interactions between the various classes as presented so far. Each class is implemented in an identically named .py file. The $__init__()$ methods intrinsic to each class are not represented in the diagram. The file math_function.py simply contains the *sigmoid*() function to compute the value of a sigmoid as defined in (1.2). Lists were used throughout the implementation since they allow for dynamic network construction. Future improvements may include adding or removing cells during training.

Given the repeated use of functions over multiple variables, vectorization could have been beneficial (via numpy, for example) but lacks flexibility. A parallel computation implementation may be considered in the future.

2.3.2 Gradient Calculation

The sensitivity implementation done during N7's Long Project was not reused. Sensitivity calculations were implemented within each corresponding layer, aiming to achieve functionality similar to the feedforward pass for backpropagation. Each network layer calls the subsequent layer to propagate the gradient calculation. Sensitivity calculation is implemented in each layer's *activate()* method and

can be toggled on or off with a boolean argument. A new *back_propagation()* method was added in each layer to calculate the gradient.

To ensure the gradient calculation method was correctly implemented, a numerical gradient calculation was also conducted using the rate of increase. The algorithm is not detailed here, as it is auxiliary to this project. A relative difference below 10^{-4} was observed compared to the numerical calculation, indicating the analytical gradient implementation was accurate.

2.3.3 Algorithm Presentation

Each network component was reimplemented in separate files to improve clarity and reusability of functions and classes when applicable across different parts of the code. The overall structure of layers and cells is implemented as nested classes. The cell class has only an *activate()* method that computes the cell's output based on its function (fuzzification, conjunction, etc.).

Each layer has two methods : *activate()* and *back_propagation()*, which respectively compute a layer's value recursively based on the input score and the gradient of this layer. A layer comprises an association of cell classes. Each layer is designed to be adaptable to input and output data size, with the list structure maintained as previously explained. Finally, a boolean *train* parameter allows the calculation of sensitivity as required during training.

2.4 Conclusion

In this chapter, we presented the revisions made to the code from N7's Long Project. Minor theoretical adjustments were made, and the code structure and organization were revised to make better use of Object-Oriented Programming, enhancing adaptability. The data format was also revised, and a new dataset was introduced due to difficulties in separating classes in the initial dataset.

Chapitre 3

Training Algorithm

Training in this network is organized around two main objectives : optimization of numerical weights and biases, and optimization of boolean weights. The normalization layer will not be modified during training, as it merely formats the output as a probability distribution.

A description of file relationships is shown in Figure 3.1, presenting the code structure discussed in this chapter. A new class, *TrainingTree*, was implemented to manage the files handling the hybrid training mechanisms of the network. The network structure is built only during numerical optimization, *optim_num*. Boolean optimization, *optim_bool*, merely uses information relayed during numerical optimization of the various networks. How we handle the storage of this information will be discussed at the end of the chapter. The *converter.py* file also contains two functions used during information storage. Finally, the *rzo_ close()* method enables exploration of different network configurations and will be explained in this chapter.

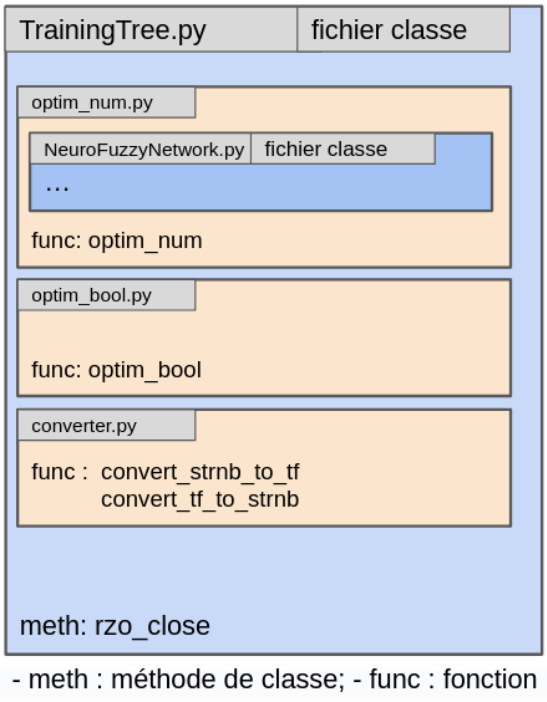


FIGURE 3.1 – Class Relationship Diagram for the Training Component

As the current network is still a prototype, we use a classic gradient descent method with a fixed step size during numerical optimization. The focus is not yet on the algorithm's efficiency but rather

on its feasibility. The cost function used is cross-entropy without a penalty term, given by equation 1.5.

3.1 Boolean Optimization

Boolean weight optimization involves selecting, from a list of potential networks, the one that will yield the best result based on projections. The pseudo-code for this algorithm is shown in Algorithm 3. For each network, we estimate the current slope of the CE, and the *PrevVal()* function uses linear projection to estimate the value it would reach after all iterations allocated to the algorithm are utilized, assuming all remaining iterations are dedicated to this network.

Algorithme 3 : Boolean Weight Optimization : Finding the Most Promising Network (OptimBool)

```
Input : rzo_dict - A dictionary containing all previously considered network configurations
        and relevant information, including the network's current CE slope and current CE
        value (CE_actuelle).
Input : iter_rest - the number of remaining iterations for training
Output : optimal_id - the most promising network
min_esp_val ← inf;
min_esp_abs ← inf;
val_pos ← True;
for rzo ∈ rzo_dict do
    val_esp ← PrevVal(rzo.CE_actuelle, rzo.pente, iter_rest);
    if val_esp < 0 then
        val_pos ← False;
        abs_esp ← PrevAbs(rzo.CE_actuelle, rzo.pente);
        if abs_esp < min_abs_esp then
            optimal_id ← id_rzo;
            min_abs_esp ← abs_esp;
        end
    end
    if val_pos And val_esp < min_esp_val then
        optimal_id ← id_rzo;
        min_esp_val ← val_pos;
    end
end
```

The average slope is calculated as a floating-point estimate : at any given time, the slope is estimated as the average over a certain number of recent iterations. This approach has several advantages :

- Limits and anticipates the required storage size, unlike an adaptive size that would require storing more and more information.
- Reduces the influence of small-scale fluctuations : it smooths out curve irregularities to best represent the trend.
- Reduces the impact of long-term fluctuations : focuses on the current performance of a network in reducing the cost function, thus limiting the influence of sudden but brief performance changes, commonly seen early in network optimization.

As previously explained, a network's performance is measured by projecting the CE over the remaining number of iterations, not indefinitely, so it's necessary to estimate the attainable value rather than just relying on the slope.

The estimated value is obtained through a linear estimation using the *PrevVal()* function. Since CE cannot be negative, two cases are considered when selecting the best network as per Algorithm 3 :

— If the network predicts a negative value, we consider the number of iterations needed to nullify the cross-entropy, calculated by *PrevAbs()*.

— If the network predicts a positive value, this value is used to compare with other networks.

A network projecting a negative value is always considered more promising than one projecting a positive value. This rule is enforced by the boolean *val_pos*, which is set to False whenever a negative CE value is projected, preventing execution of the code under the second **if** test.

To explore new networks, we use an algorithm, described in pseudo-code in Algorithm 4. When a network is optimized numerically, we also check if its neighbors have been considered. A network B is a neighbor of network R if :

$$\exists!\ (b_R, b_B) \in W_R^{bool} \times W_B^{bool}, b_R \sim b_B | b_R \neq b_B$$

, where $\sim$ means the weights are in the same position in the network, and W_R^{bool} corresponds to the set of weights in the neuro-fuzzy network R.

If a network's neighbors have not been considered, the network is closed by iterating through all boolean weights and generating a new configuration using the $Switch()$ function, which reverses a network weight. If the network has not been considered before, the $OptimNum()$ function estimates the value achieved by the new network over a small number of iterations by optimizing it via back-propagation. We then perform boolean optimization as described above on all neighbors stored in the *voisin_dict* (dictionary in Python terminology). Only the most promising network, *best_voisin*, is added to our network dictionary.

Only one neighbor is added because the number of networks can grow exponentially (there are $2^{number\_of\_boolean\_weights}$ potential networks). To limit the number of neighbors considered, a tabu method is used : each network weight is assigned a seniority value, representing the number of open networks since that weight was modified. This approach also promotes exploration by forcing the algorithm to explore paths it might not have taken otherwise due to being suboptimal compared to all choices at each stage.

While considering potential neighbors, a special case applies to the last layer of boolean weights. To predict a class, at least one weight must be set to True. Thus, if only one weight is True in the final layer, we don't consider the neighboring network that would set this weight to False, rendering the network unusable.

Finally, the $PositionCritique()$ condition is defined as follows :

— The seniority criterion is not met.

— The weight is in a critical position on the last layer.

— The configuration already exists in the set of considered networks.

The $Converter()$ function assigns each network a unique identifier based on its boolean weight configuration. This method will be detailed in the next section of this chapter.

3.2 Overall Network Optimization Process

To optimize this mixed-weight network, we alternate between boolean weight optimization phases and numeric weight and bias optimization phases. The decision tree at a given time t in the training

Algorithme 4 : Closing a Network (FermetureRzo)

Input : rzo_parent - A dictionary containing all information of the network from which we are exploring neighbors.
Input : iter_rest - the remaining number of iterations
Output : best_voisin - the most promising neighboring network

```
poids_bools ← rzo_parent.poids_bools;
ancienneté ← rzo_parent.ancienneté;
for bool ∈ poids_bools do
    if not(PositionCritique(bool, ancienneté)) then
        voisin_bools ← Switch(bool);
        id_voisin ← Converter(voisin_bools);
        rzo_voisin ← OptimNum(rzo_parent, voisin_bools, iter_rest);
        voisin_dict[id_voisin] ← rzo_voisin;
    end
end
best_voisin ← OptimBool(voisin_dict, iter_rest);
```

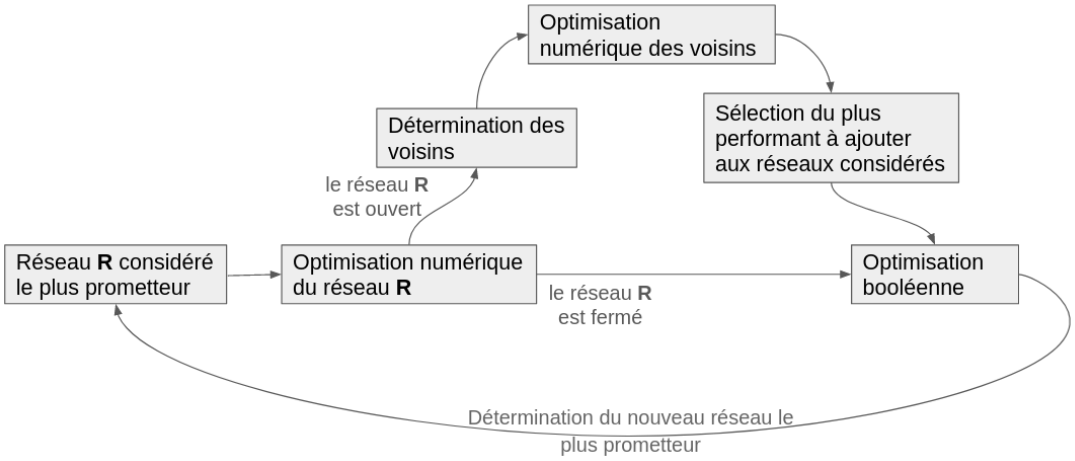


FIGURE 3.2 – Description of the Learning Algorithm for Continuous and Boolean Variables

process is shown in Figure 3.2. Let R be a network currently considered the most promising based on the remaining time and configuration. We start by optimizing its numerical weights and biases over a fixed number *nbrep* of iterations using a backpropagation algorithm. Once completed, if the network is open, we close it using the $FermetureRzo()$ function. Otherwise, we optimize the boolean weights by selecting as the new network, $meilleur\_rzo\_actuel$, the one that yields the most promising result. The complete pseudo-code is presented in Algorithm 5.

NOTE : We start by optimizing numeric weights and biases before boolean weights to maintain coherence between the initial network and all those considered by neighborhood evolution.

3.3 Storage

To efficiently switch from one potential network to another, various network-specific details need to be stored. For this purpose, we establish a nested dictionary as shown in Figure 3.4. Each sub-dictionary is identified by a unique key, corresponding to the decimal conversion of concatenated

Algorithme 5 : Training with Boolean and Continuous Variables

```
Input : rzo - A mixed-weight neuro-fuzzy network
Input : iter_dispo - the total number of possible backpropagation steps
Input : nbrep - the number of backpropagation iterations for each numeric weight
        optimization
Input : data - the training dataset to be modeled.
Output : meilleur_rzo_actuel : the most promising neuro-fuzzy network currently, with
         adjusted parameters
id_rzo ← Converter(rzo.poids_bools);
dict_rzo[id_rzo] ← rzo;
meilleur_rzo_actuel ← rzo;
while iter_dispo > 0 do
    meilleur_rzo_actuel.OptimNum(data, nbrep);
    iter_dispo ← iter_dispo - bp_pas;
    if meilleur_rzo_actuel.estOuvert then
        meilleur_voisin ← FermetureRzo(meilleur_rzo_actuel, iter_disp);
        id_meilleur_voisin ← Converter(meilleur_voisin);
        dict_rzo[id_meilleur_voisin] ← meilleur_voisin);
        meilleur_rzo_actuel.estOuvert ← False;
    end
    meilleur_rzo_actuel ← Optimbool(rzo_dict, iter_disp);
end
```

boolean weights in binary format. A more concrete explanation of this process is available in Figure 3.3. The functions *convert_strnb_to_tf*() and *convert_tf_to_strnb*() convert between decimal and boolean representations.

Below is a description of the information stored in each sub-dictionary :

- poids_num : List of the network's numeric weights in its last state. This value is initialized with the parent network's weights upon opening.
- biais_num : List of the network's numeric biases in its last state, initialized analogously to weights_num.
- ce_evol : Floating-point list of the most recent cross-entropy values obtained during the network's numeric optimization, used to estimate the network's CE slope.
- pente : Slope estimate of the network's CE.
- estOuvert : Network state, *False* if the network's neighbors have been considered, *True* otherwise.
- bool_seniority : List of all boolean weight seniority values needed when opening the network. This list is defined based on the parent network's values, setting the seniority of modified weights to 0 and incrementing the rest by 1. This value is no longer considered once the network is closed.

This storage method ensures only one active network at a time and allows for efficient activation and deactivation of networks without data loss.

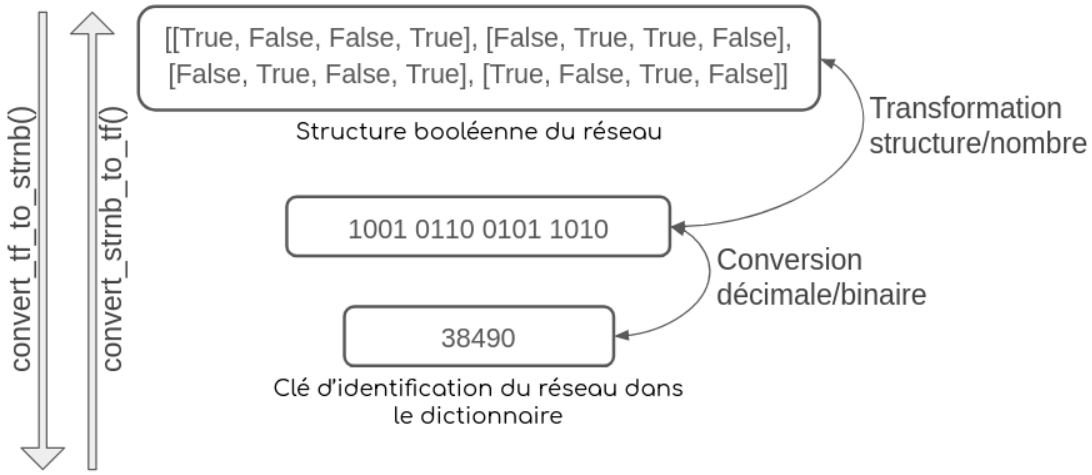


FIGURE 3.3 – Mechanism for Key Assignment during Storage

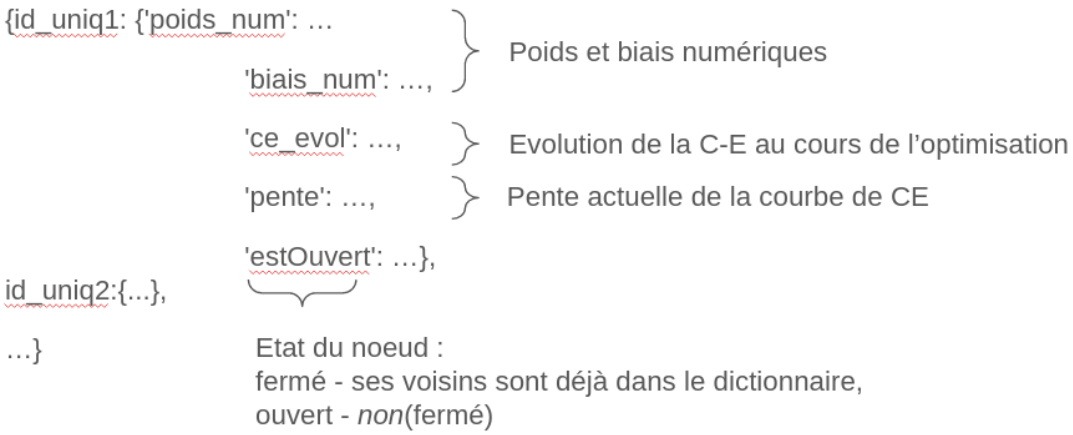


FIGURE 3.4 – Visualization of Network Storage Method

3.4 Conclusion

In this chapter, we provided a detailed explanation of the optimization method developed for a neural network with both boolean and continuous variables. This method alternates between two types of optimization : - Local and monotonic numeric optimization, only refining the best network ; - Boolean weight optimization, also local but non-monotonic (allowing a neighbor that may be less optimal than the current solution) through selecting the optimal network among all considered networks.

We also outlined how networks are stored during training.

Chapitre 4

Establishing the Decision Rule

To make the network interpretable after training, multiple transformations are applied to translate its various numeric and boolean values.

4.1 Linguistic Transcription

The first transformation involves naively translating all weights and biases.

Numeric Weights and Biases : Given that our fuzzification function is a sigmoid function (as defined in 1.1), a simple calculation shows that the quotient $-\frac{b}{w}$ gives the x-axis value where the sigmoid equals 0.5, with w being the numeric weight and b the associated bias. In our case, we consider only the descriptors *small* and *large* for each score. Thus, in the case of the *small* descriptor, values below $-\frac{b}{w}$ are considered small, and for *large*, values above this quotient are considered large.

Boolean Weights : Boolean weights are translated by the logical meaning of each layer. Thus, the rule for a given class will correspond to a disjunction of conjunctions. For a given class i, we examine the boolean weights $w_{i,j}^{disj}$. For each non-zero weight, we define an element of the disjunction as part of the rule for this class. This element is further specified by examining the weights $w_{i,j}^{conj}$ for the corresponding i. The resulting conjunction corresponds to one of the elements of the disjunction. A file, desc_data.py, contains a dictionary with information on each score and its position in the network structure to translate the various non-zero booleans into words. The descriptor, *small* or *large*, is determined by the direction of the slope.

4.2 Rule Simplification

To make the rule practically usable, it must be simplified after the previous operation. The boolean weight exploration method is not intelligent and simply adds and removes rules without verifying whether parts of the new configuration become obsolete. This results in a significant aggregation of rules, making it difficult to derive meaningful rules. After applying the algorithm described below, an average of two scores are used per rule, as opposed to seven before simplification.

The *clean_useless()* function, applied to a given network, simplifies it. Algorithm 6 provides its functioning in pseudo-code.

For a given network, we create $rzo\_simplif$, a copy of this network by value. We go through all the boolean weights in $rzo\_simplif.poids\_bools$, setting each active weight to False. The cross-entropy of the resulting network is calculated by the $Cost()$ function and compared to ce_ini, the

Algorithme 6 : Simplifying the Decision Rule

```
Input : rzo - A mixed-weight neuro-fuzzy network
Input : data - the training dataset to be modeled.
Output : simplif_rzo : an equivalent neuro-fuzzy network with maximized boolean weights
         set to False
rzo_simplif ← Copy(rzo);
ce_ini ← Cost(rzo, data);
reverse_poids ← Reverse(rzo.poids_bools) ;
for bool ∈ reverse_poids do
    if bool = True then
        rzo_simplif.poids_bools ← Switch(bool, rzo_simplif.poids_bools);
        ce_simplif ← Cost(rzo_simplif, data);
        if ce_ini ≠ ce_simplif then
            bool_simplif ← not(bool);
            rzo_simplif.poids_bools ← Switch(bool_simplif, rzo.poids_bools);
        end
    end
end
```

cross-entropy of the initial network. If no difference is observed, we keep the weight deactivated ; otherwise, we reactivate it using $Switch()$.

We reverse the layer traversal order using the $Reverse()$ function, starting with the layers closest to the output. Layers are inversely prioritized in the rule structure relative to their position in the network. Thus, a disjunction can be set to False, making an entire conjunction obsolete. Conversely, setting a conjunction to False provides no information on a disjunction.

Also, the rule simplification is not unique, as two parameters providing the same information allow for the simplification of one or the other, leading to two different final rules. However, given the uniform simplification method and the boolean traversal order, one rule is always favored, ensuring that two networks with the same simplification potential do not end up with different simplifications.

4.3 Conclusion

In this chapter, we presented the strategy used to make a post-training network intelligible and understandable to humans. This strategy first involves translating the various weight and bias values, followed by intelligent rule simplification.

Chapitre 5

Results

In this chapter, we apply the network we constructed on a standard simulation, using parameters determined empirically during development to observe the algorithm's behavior. We then vary each parameter to measure its influence on algorithmic function and performance. Finally, we discuss potential improvement avenues that could be explored.

5.1 Output Archiving

Significant effort has been devoted to automatically archiving all information generated during the program's execution to keep track of everything accomplished. Each program run is therefore saved, providing several insights :

— Visualization of CE evolution for networks considered during each numeric optimization, distinguished by color.

— Visualization of membership functions of the network after optimization for all scores in a given dataset.

— Creation of a text file recording execution time, number of networks opened, complete program configuration during execution, final weight and bias values, cross-entropy values, confusion matrix on both the training and validation datasets, and finally the boolean weights before and after simplification.

The *confusion matrix* here refers to the average probability matrix of the network's predictions relative to actual output.

5.2 Standard Execution Configuration

For our various tests, we use the "Breast Cancer Wisconsin (Diagnostic)" dataset. The standard configuration is presented in Table 5.1. These values were defined empirically during program implementation.

NBREP	10
EPSILON	1
ITER_GLOB	20000
SENIORITY	46
CE_FOR_SLOPE	5

Table 5.1 – Initial Configuration of Global Parameters

Input parameters are :

— Initial weights and biases : These represent the initial network variables and reflect expert knowledge, assumed to be close to reality.

— NBREP : Number of consecutive iterations performed in numeric optimization (backpropagation), for either the most promising network or one of its neighbors.

— EPSILON : Step size in the gradient descent algorithm (constant step size used).

— ITER_GLOB : Maximum gradient iterations for all networks ; serves as the stop criterion. Iterations used during neighbor optimization to select the most relevant network are also counted since they are not negligible in the overall optimization.

— SENIORITY : Maximum seniority threshold after which a boolean weight can be modified again.

— CE_FOR_SLOPE : Buffer size for estimating CE slope, where the number of iterations between extreme values equals $CE\_for\_slope * Nbrep$.

5.2.1 Initial Rule

An initial expert rule must be defined for our data. Based on the visualization of various data shown in Figure 2.1, we can approximately define boundaries so that each class is predominant on either side. Boolean weights can also be defined based on this observation. These values are presented in Table 5.2, and boolean weight values in Table 5.3.

	Radius_m	Texture_m	Perimeter_m	Area_m	Concavity_m
w *high*	1.0	1.0	1.0	1.0	1.0
b *high*	-0.5	-0.5	-0.5	-0.3	-0.2
w *low*	-1.0	-1.0	-1.0	-1.0	-1.0
b *low*	0.5	0.5	0.5	0.3	0.2

TABLE 5.2 – Standard Initial Network Weight and Bias Values

Cell 1	True	False	True	False	True	False	True	False	True	False
Cell 2	False	True	False	True	False	True	False	True	False	True
Cell 3	False	False	False	False	False	False	False	False	False	False
Cell 4	False	False	False	False	False	False	False	False	False	False

(a) Conjunction Layer

Cell 1	False	True	False	False
Cell 2	True	False	False	False

(b) Disjunction Layer

TABLE 5.3 – Standard Initial Boolean Weight Values of the Network

Thus, the initial rules for the two classes are :

— Predict B if Radius_m is low (< 0.5), Texture_m is low (< 0.5), Perimeter_m is low (< 0.5), Area_m is low (< 0.3), and Concavity_m is low (< 0.2).

— Predict M if Radius_m is high (> 0.5), Texture_m is high (> 0.5), Perimeter_m is high (> 0.5), Area_m is high (> 0.3), and Concavity_m is high (> 0.2).

It is worth noting that Cells 3 and 4 of the conjunction layer are not used in the initial network, allowing additional rules to be added during training.

5.2.2 First Result with the Algorithm and Standard Configuration

Before evaluating each global parameter's influence and refining them, we ran the program with standard parameters to establish a baseline for comparison.

Table 5.4 presents the numeric results from this training. Although assessing computation time may be premature since speed optimization is not the current focus, it is noteworthy that the time is not overly long. Few networks are opened relative to the potential number of networks (there are $2^{boolean\_variable\_count}$ possible boolean weight configurations). Notably, cross-entropy (CE) decreases during training, and similar CE values between training and validation datasets indicate a lack of overfitting.

Execution Time	36'31"
Closed Networks	33
Initial Network CE	0.6527
Training Data CE	0.2184
Validation Data CE	0.1882

TABLE 5.4 – Algorithm Execution Results in Standard Configuration

The decision rule determined after training is :

— Predict B if Radius_m is low ($<$ -0.78).

— Predict M if Concavity_m is low ($<$ -2.89).

This rule diverges significantly from our initial intuition. Moreover, it contradicts our expectations since class M is predicted with the qualifier "low" where it should logically be predicted as "high" for the Concavity_m score (see Figure 2.1).

Figure 5.1 illustrates an undesired behavior of the algorithm. Ideally, criteria used in decision rules should cover a broad range in $[0; 1]$. However, the two criteria used in these rules exhibit extremely low coverage across this interval.

This issue may be due to the final network normalization layer. As this layer calculates ratios of the preceding layer's outputs, there is no constraint

5.3 Optimization of Input Parameters

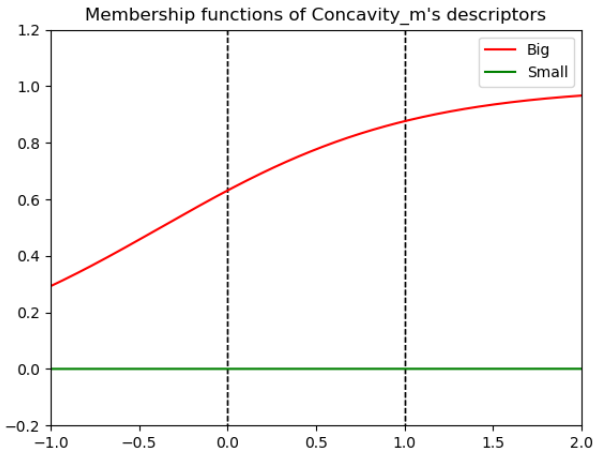
Next, we vary the global parameters to determine their influences and test hypotheses we proposed. Testing limit configurations allows us to ensure the algorithm functions correctly in "extreme" cases (very small or very large values) and verifies its robustness.

5.3.1 Initial Weights and Biases

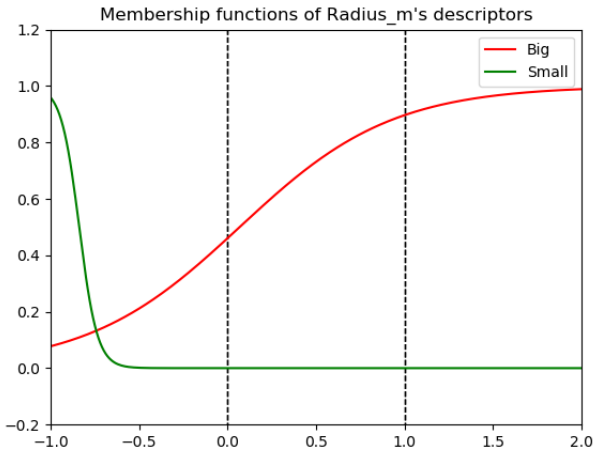
To assess the impact of the expert rule, we implemented a function, *rd_weights()*, which randomizes either the numeric weights and biases or the boolean weights while retaining the desired structure. Two experiments were conducted : first, we varied the numeric weights and biases, and second, we varied only the boolean weights.

Randomization of All Numeric Variables

In this experiment, the signs of the weights and biases were fixed. Changing the sign would alter the direction of descent for the associated sigmoid, thereby switching from a *small* descriptor to a *large* one, or vice versa, which we wanted to avoid to maintain consistency. Additionally, descriptor



(a) Score Concavity_m



(b) Score Radius_m

FIGURE 5.1 – Visualization of Membership Functions of Scores Used for Decision Rules in Standard Configuration

type changes are addressed in the next experiment. Although not shown here in full, the results can be accessed online. Currently, this test yielded no relevant findings due to the program's limitations.

Notably, the number of networks considered during learning varied widely, from 18 to 41, and CE on validation data ranged from 0.16 to 0.35. The final rules learned varied significantly across configurations, revealing no discernible pattern.

Randomization of Boolean Weights Only

In this experiment, only the boolean weights were randomized, while the initial numeric weights were set to average values ($w = \pm 1$, $b = \mp 0.5$) to represent moderate slopes and symmetric membership functions on $[0, 1]$. Numeric weights were not in their standard configuration (see Table 5.2) due to unpredictable connections with randomized boolean weights. As with the first experiment, this test revealed no clear trends. CE on validation data ranged from 0.20 to 0.31, with the number of opened networks varying from 22 to 61. A wide range of rules was observed as well.

Conclusion

We can partially conclude that the initial weight and bias values influence both performance (in terms of accuracy) and the final rule established post-learning. Moreover, the so-called *expert* rule (see Table 5.1) did not outperform the randomized rules in accuracy. However, its performance was relatively good across randomized trials. Evaluating rule validity remains difficult as all rules thus far have proven incorrect, preventing us from identifying relevant program behaviors based on these experiments.

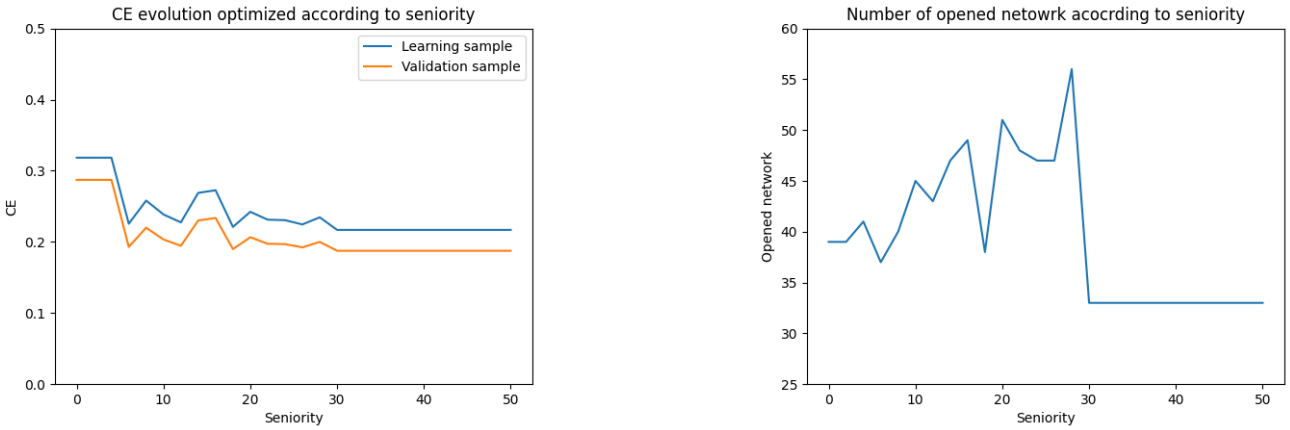
5.3.2 Seniority Criterion (*SENIORITY*)

To assess the impact of the seniority criterion in the tabu method, we varied it from 0 to the maximum, which is the number of boolean weights in the network. Setting it to zero corresponds to not using the tabu method at all, while setting it to the maximum limits the network to modifying only one weight at a time in a stable state, effectively restricting weight modification to one per network iteration.

Since the seniority criterion forces the network to take suboptimal trajectories, we anticipated an increase in the number of networks as seniority rises. This criterion also allows the network to escape local minima, so we expected improved performance and better rule formation over time.

Experiments

Results are shown in Figure 5.2. We observe a noticeable improvement in algorithm performance with the use of the tabu method. Performance stabilizes beyond a seniority value of 30, likely because further increases in seniority prevent networks from deviating sufficiently to alter the network's path, leading to repeated trajectories.



(a) Evolution of CE after Optimization

(b) Number of Networks Considered

FIGURE 5.2 – Results After Varying the Seniority Criterion

Generally, the decision rule can be summarized as :

— Predict B if Radius_m is low ($<$ -0.78).

— Predict M if Concavity_m is low ($<$ -2.89).

This rule is challenging to interpret and closely resembles that obtained for the standard configuration, differing only slightly in threshold values. Similar issues were found regardless of the seniority criterion value.

A notable difference appeared for seniority values below 4, yielding different rules without clear justification, although they were not more coherent.

Conclusions

Our initial assumptions were largely validated : increasing the seniority criterion improved CE and increased the number of opened networks. However, excessive seniority impaired exploration after a threshold of 30, as explained. An optimal balance is crucial to minimize CE after learning without sacrificing exploration.

5.3.3 Global Iterations (*ITER_GLOB*)

To understand this criterion's impact, we conducted the same experiment with different iteration counts : 5000, 20000, 50000, 200000, and 500000. This allowed us to examine the algorithm's behavior over extended periods, its sensitivity to iteration increases, and potential overfitting.

Experiments

Results are shown in Figure 5.3. We see a clear improvement in CE and the number of networks considered in initial experiments, with negligible differences between 200000 and 500000 iterations. Computation time (linearly dependent on iteration count) increased significantly, making excessive iterations unnecessary.

With only 5000 iterations, learning was insufficient for the data and task complexity. At 20000 iterations, results were typical, while 50000 iterations produced this rule :

— Predict B if Area_m is low ($<$ -0.01).

— Predict M if (Radius_m is low ($<$ -0.68)) or (Concavity_m is high ($>$ 0.56) and Concavity_m is low ($<$ -0.32)).

Besides earlier-mentioned issues (see Annex B.1), we observe contradictory rules that cannot be managed by the algorithm. This could indicate an erroneous rule, a need for a *medium* descriptor, or an element that can simultaneously be both high and low. Notably, no overfitting was detected, even with large iteration counts.

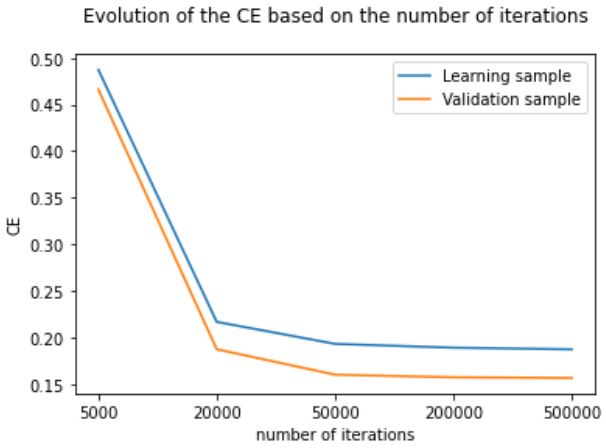
NOTE : In Figure 5.3c, the x-axis is not linear.

Conclusion

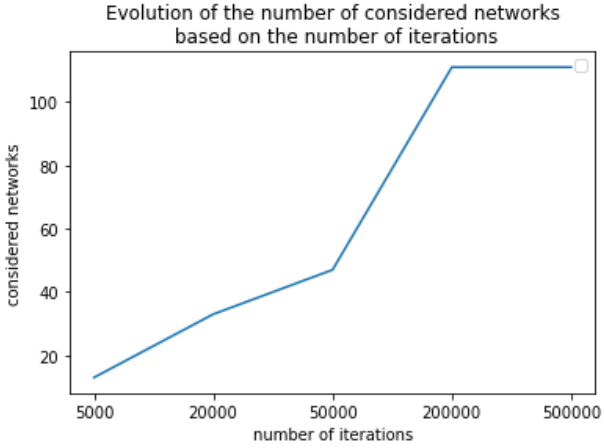
This experiment indicates that excessive iteration counts are not useful for this algorithm in its current form. Beyond a certain point, neither exploitation nor exploration improves, and contradictory rules appear, which the algorithm cannot handle. Nonetheless, robustness to overfitting was confirmed.

5.3.4 Iterations in Numeric Optimization (*NBREP*)

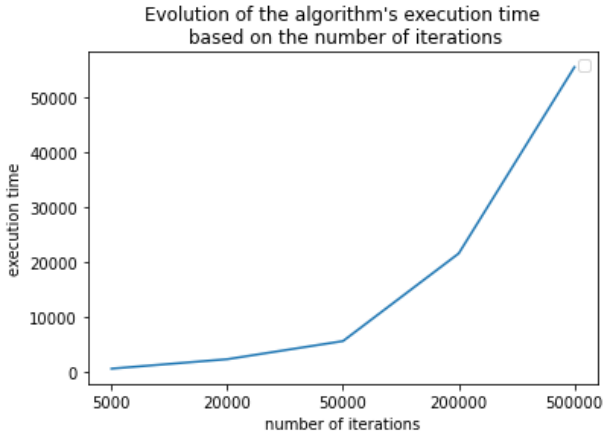
To assess the influence of this criterion, we tested five values : 2, 10, 50, and 100. We expect higher values to favor exploitation over exploration, while very low values (2) could make the algorithm inefficient due to frequent network changes. Because CE slope is linked to this parameter, lower values make the algorithm more sensitive to minor variations.



(a) CE After Optimization by Total Iterations



(b) Number of Networks Considered



(c) Execution Time

FIGURE 5.3 – Results After Varying Total Iterations

Experiment

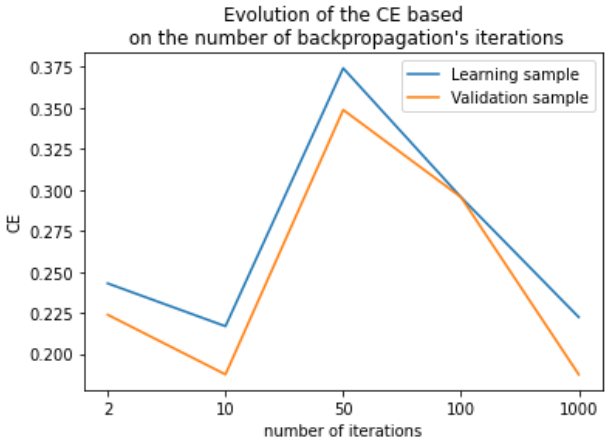
Results are shown in Figure 5.4. Significant CE variation is seen in Figure 5.4a. Interestingly, extremely high iteration counts (1000) produce strong CE reduction, but as shown in Figure 5.4b, only two networks were explored, indicating that exploitation fully overshadowed exploration. Thus, excessively high values are counterproductive. Generally, increased backpropagation iterations reduce exploration since the same iterations are allocated to each neighbor during opening.

Conclusion

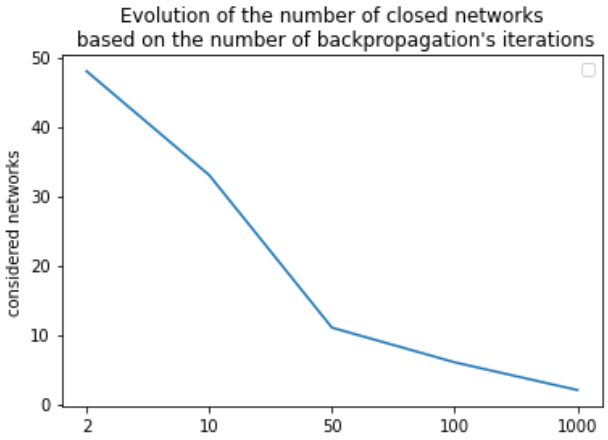
A balance is necessary to support both exploration and exploitation. Large values appear beneficial based solely on CE evolution, but Figure 5.4b shows that exploration is entirely neglected in such cases.

5.3.5 Buffer Size (*CE_FOR_SLOPE*)

To understand the influence of this criterion, we varied it across five values : 2, 5, 10, 20, and 50. A value of 2 considers only the latest numeric optimization (plus the most recent numeric optimization result) to calculate the slope, effectively treating it as a minimal buffer. In this case, we expect high



(a) CE After Optimization



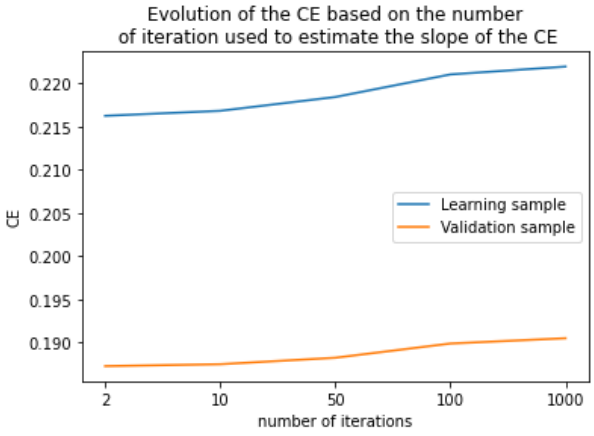
(b) Number of Networks Considered

FIGURE 5.4 – Results After Varying Iterations in Numeric Optimization

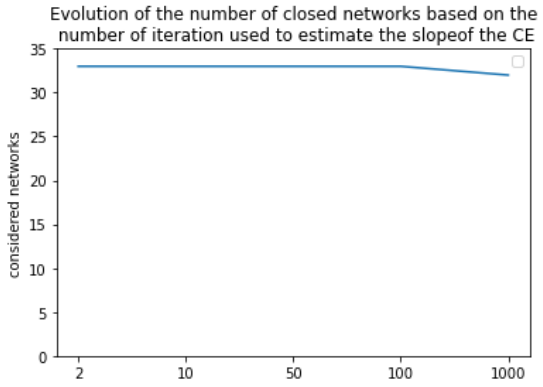
sensitivity to minor fluctuations. Conversely, a very high value (50) would mean using nearly all values for most networks to estimate the slope. From earlier observations, we can infer that most networks won't undergo 50 rounds of numeric optimization, so networks with initially high CE will be favored, as they generally have a steep slope at the outset.

Experiments

The results are presented in Figure 5.5. We observed very minimal differences in both exploitation and exploration. The decision rule and descriptor thresholds remained consistent across variations. However, a difference was seen in the exploitation of potential networks : with increasing iteration counts, there was a slight preference for networks with strong initial improvements. Figures for each experiment are available in Appendix B.3.



(a) Evolution of CE



(b) Evolution of the Number of Considered Networks

FIGURE 5.5 – Results After Varying the Buffer Size for CE Slope Estimation

Conclusion

This parameter appears to have little effect on algorithm outcomes. One possible reason for this lack of sensitivity could lie in the general trend of CE, as seen with EPSILON = 1 (see Appendix B.2 for an example). The CE trend is sufficiently regular that no abrupt changes significantly alter the estimated slope based on buffer size. Nevertheless, we noted that with a larger buffer, networks with strong initial gains tended to be favored. Hence, selecting a moderate value seems best to avoid this bias while not being overly responsive to small fluctuations with a smaller buffer size.

5.4 Improvement Paths

As explained, this network is still in the prototype stage, leaving room for many improvements. Several issues remain unresolved ; currently, the program cannot meet our requirements for producing human-justifiable rules. Implementing constraints to control output-space coverage by membership functions is one avenue for addressing this, with the goal of producing coherent rules and thresholds. Another consideration is managing contradictory rules that can arise in the decision process.

Other optimizations include adjusting iteration allocations during numeric optimization based on whether they occur during exploration of new networks or exploitation of existing ones. Enhanced gradient descent techniques or adaptive step sizes, such as a Wolfe step [22], could also accelerate convergence.

Parallelization could also improve speed, given the algorithm's repetitive tasks—matrix calculations, identical algorithms applied to various scores, exploration of neighbors, etc. Additionally, it could be beneficial to allow the network to add or remove linguistic descriptors if those provided are insufficient. Finally, a comprehensive, human-readable standard could be developed to allow for tracking decision-making processes during learning.

5.5 Conclusion

In this chapter, we observed the functioning of our algorithm. While it does not currently produce rules that align with observations, this is not problematic ; the algorithm is still in development, and identified issues are being addressed in subsequent versions. We also conducted single-parameter comparisons of the network's global parameters to determine their influences and examine algorithm responses to various conditions. Following this, we proposed multiple improvement paths.

Chapitre 6

General Conclusions

6.1 Subject Conclusion

We observed that explainability is an increasingly sought-after component in learning algorithms, though no existing solution completely addresses the issue. Neuro-fuzzy neural networks appear promising, and many researchers are exploring their use. The neuro-fuzzy network model developed here is a simple four-layer perceptron with both boolean and continuous parameters. Its simplicity is intentional, as our objective is to examine the feasibility and relevance of this network type.

A tailored learning method was developed and implemented for this hybrid network, alternating between numeric optimization via gradient descent and backpropagation, and boolean optimization by creating a configuration tree of network states. Although this project is ongoing, intermediate results, while not yet producing intelligible rules, are encouraging. The learning phase and boolean weight optimization function as expected. Problems impacting algorithm operation were identified—such as poor output-space coverage by membership functions—and proposed solutions are currently being evaluated in the next stage of my internship.

6.2 Personal Experience

Throughout this internship, I gained insight into the research field. I also broadened my mathematical knowledge in areas not specifically covered in the INSA GMM track, including set theory, fuzzy logic, and discrete optimization. Additionally, I improved my programming skills to develop a usable and adaptable product.

Since this theoretical project is in its early stages, each newly identified issue or idea opens up multiple directions. As a result, I needed to adopt strong organizational skills and effective methodology to stay focused on parallel ideas and tracks. This involved consistently organizing code, meticulously archiving versions, and considering result archiving as a critical component of development. I also kept a project journal, detailing ongoing experiments, code modifications, meeting notes, and ideas.

I was able to immediately engage with the subject matter and make progress quickly. In contrast, report writing posed the most significant challenge, revealing a need for improvement if I intend to pursue a PhD after completing my engineering program. My challenge lay primarily in maintaining focus on one section without getting sidetracked by others.

Bibliographie

[1] Cardon Dominique, Cointet Jean-Philippe, and Mazières Antoine. La revanche des neurones. l'invention des machines inductives et la controverse de l'intelligence artificielle. *Réseaux*, 211(5) :173–220, 2018.

[2] Ashok Goel. Looking back, looking ahead : Symbolic versus connectionist ai. *AI Magazine*, 42(4) :83–85, 2022.

[3] Paul Smolensky. Connectionist ai, symbolic ai, and the brain. *Artificial Intelligence Review*, 1(2) :95–109, 1987.

[4] Béatrice Laurent, Philippe Besse, and Mélisande Albert. Neural networks and introduction to deeplearning, 2022-2023.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088) :533–536, 1986.

[6] Philippe Besse, Eustasio del Barrio, Paula Gordaliza, Jean-Michel Loubes, and Laurent Risser. A survey of bias in machine learning through the prism of statistical parity for the adult data set, 2020.

[7] P. Lee, M. Le Saux, R. Siegel, M. Goyal, C. Chen, and Y. Meltzer Ma. Racial and ethnic disparities in the management of acute pain in us emergency departments : Meta-analysis and systematic review. *The American journal of emergency medicine*, 37, 2019.

[8] Alexei Grinbaum, Raja Chatila, Laurence Devillers, Caroline Martin, Claude Kirchner, Jérôme Perrin, and Catherine Tessier. Systèmes d'intelligence artificielle générative : enjeux d'éthique. Technical report, Comité national pilote d'éthique du numérique, July 2023.

[9] Mélanie Gornet and Winston Maxwell. Normes techniques et éthique de l'IA. In *Conférence Nationale en Intelligence Artificielle (CNIA)*, Strasbourg, France, July 2023.

[10] CNIL. Les enjeux éthiques des algorithmes et de l'intelligence artificielle, synthÈse du dÉbat public animÉ par la cnil dans le cadre de la mission de rÉflexion Éthique confiÉe par la loi pour une rÉpublique numÉrique, 2017.

[11] Commission européenne. RÈglement du parlement europÉen et du conseilÉtablissant des rÈgles harmonisÉes concernant l'intelligence artificielle (lÉgislation sur l'intelligence artificielle) et modifiant certains actes lÉgislatifs de l'union, 2021.

[12] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3) :338–353, 1965.

[13] Madan M Gupta and J11043360726 Qi. Theory of t-norms and fuzzy inference methods. *Fuzzy sets and systems*, 40(3) :431–450, 1991.

[14] J-SR Jang. Anfis : adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3) :665–685, 1993.

[15] T. Takagi and M. Sugeno. Derivation of fuzzy control rules from human operator's control actions. *IFAC Proceedings Volumes*, 16(13) :55–60, 1983. IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, Marseille, France, 19-21 July, 1983.

[16] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5) :206–215, 2019.

[17] Alan FT Winfield, Serena Booth, Louise A Dennis, Takashi Egawa, Helen Hastie, Naomi Jacobs, Roderick I Muttram, Joanna I Olszewska, Fahimeh Rajabiyazdi, Andreas Theodorou, et al. Ieee p7001 : A proposed standard on transparency. *Frontiers in Robotics and AI*, 8 :665729, 2021.

[18] KV Shihabudheen and Gopinatha N Pillai. Recent advances in neuro-fuzzy system : A survey. *Knowledge-Based Systems*, 152 :136–162, 2018.

[19] Pietro Barbiero, Gabriele Ciravegna, Francesco Giannini, Mateo Espinosa Zarlenga, Lucie Charlotte Magister, Alberto Tonda, Pietro Lio, Frederic Precioso, Mateja Jamnik, and Giuseppe Marra. Interpretable neural-symbolic concept reasoning. *arXiv preprint arXiv :2304.14068*, 2023.

[20] Jean-Loup Farges, Guillaume Infantes, Charles Lesire, and Augustin Manecy. Using pomdp with raw observations for detecting and recognizing objects of interest.

[21] William Nick Street, William H. Wolberg, and Olvi L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Electronic imaging*, 1993.

[22] Aude Rondepierre. Méthodes numériques pour l'optimisationnon linéaire déterministe, 2021-2022.

Annexe A

Data Sets

A.1 Means and Variances of the Aircraft Crash Data Without Stratified Sampling

Classe Box	moyenne	variance	taille échantilon
ScoreOB learning	0.745845	0.312802	92
ScoreOB validation	0.642614	0.123770	264
ScoreYJ learning	0.251455	0.213133	92
ScoreYJ validation	0.312802	0.247477	264

Classe Empty	moyenne	variance	taille échantilon
ScoreOB learning	0.601494	0.141876	95
ScoreOB validation	0.497156	0.195171	345
ScoreYJ learning	0.318352	0.272217	95
ScoreYJ validation	0.238996	0.237524	345

Class Jacket	moyenne	variance	taille échantilon
ScoreOB learning	0.605405	0.148332	100
ScoreOB validation	0.480519	0.181492	273
ScoreYJ learning	0.501369	0.093092	100
ScoreYJ validation	0.601748	0.192763	273

FIGURE A.1 – Means and Variances of the Aircraft Crash Data Without Stratified Sampling

A.2 Means and Variances of the Aircraft Crash Data With Stratified Sampling

Classe Box	moyenne	variance	taille échantilon
ScoreOB learning	0.669080	0.116400	249
ScoreOB validation	0.669786	0.127065	107
ScoreYJ learning	0.298989	0.238507	249
ScoreYJ validation	0.292198	0.245265	107

Classe Empty	moyenne	variance	taille échantilon
ScoreOB learning	0.518163	0.193048	308
ScoreOB validation	0.523231	0.182244	132
ScoreYJ learning	0.251838	0.243893	308
ScoreYJ validation	0.266141	0.255735	132

Class Jacket	moyenne	variance	taille échantilon
ScoreOB learning	0.508750	0.182935	261
ScoreOB validation	0.526237	0.178682	112
ScoreYJ learning	0.571727	0.177171	261
ScoreYJ validation	0.582067	0.177959	112

FIGURE A.2 – Means and Variances of the Aircraft Crash Data With Stratified Sampling

A.3 Means and Variances of the "Breast Cancer Wisconsin" Data Set

Classe B	moyenne	variance	taille échantillon
Radius_m learning	0.428148	0.063663	250
Radius_m validation	0.427991	0.059953	107
Texture_m learning	0.451582	0.101854	250
Texture_m validation	0.453218	0.099715	107
Perimeter_m learning	0.400606	0.062938	250
Perimeter_m validation	0.409209	0.058980	107
Area_m learning	0.180149	0.054990	250
Area_m validation	0.183109	0.048202	107
Concavity_m learning	0.104535	0.096772	250
Concavity_m validation	0.102442	0.110669	107

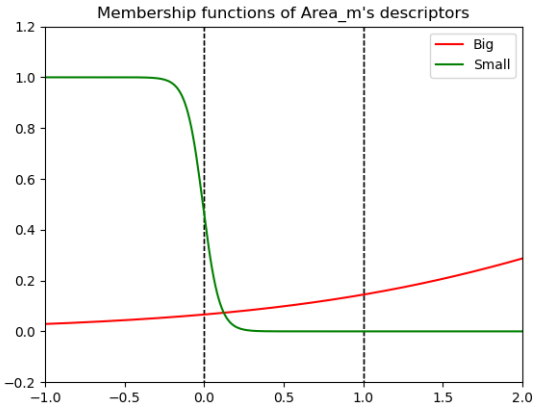
Classe M	moyenne	vairance	taille échantillon
Radius_m learning	0.616944	0.108601	148
Radius_m validation	0.631147	0.124161	64
Texture_m learning	0.546328	0.092707	148
Texture_m validation	0.558565	0.102679	64
Perimeter_m learning	0.607113	0.110494	148
Perimeter_m validation	0.623358	0.126092	64
Area_m learning	0.465174	0.141275	148
Area_m validation	0.478565	0.157871	64
Concavity_m learning	0.370676	0.169729	148
Concavity_m validation	0.390621	0.186984	64

FIGURE A.3 – Means and Variances of the "Breast Cancer Wisconsin" Data Set

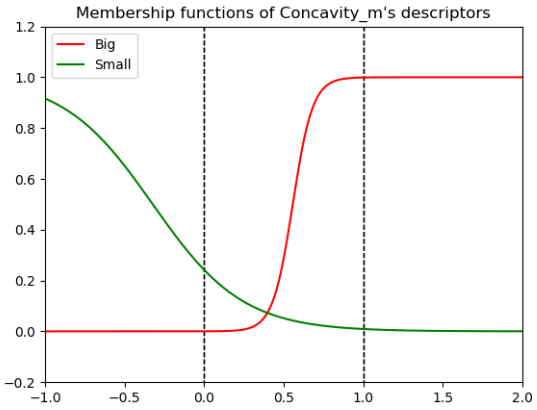
Annexe B

Results

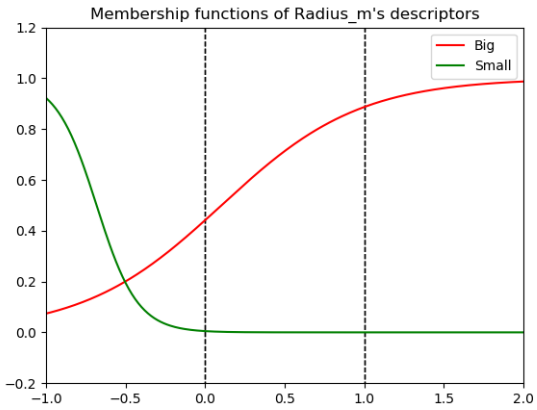
B.1 Membership Functions



(a) Score Area_m



(b) Score Concavity_m



(c) Score Radius_m

FIGURE B.1 – Membership Function for Rules Found From 50000 Iterations

B.2 Typical CE Profile

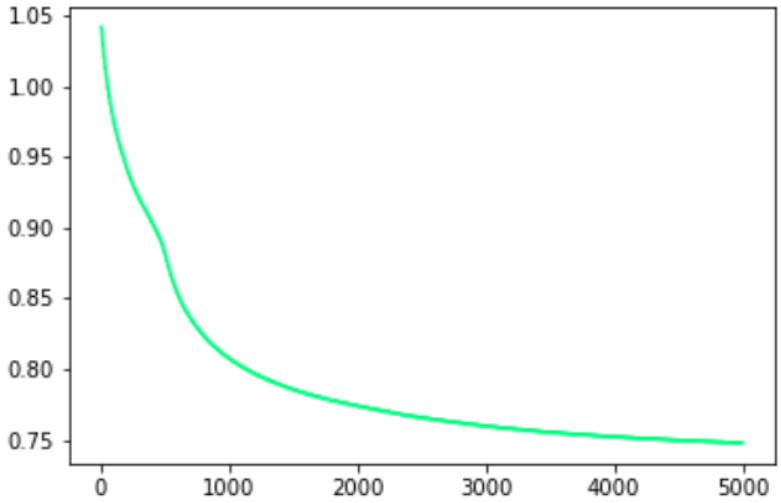


FIGURE B.2 – Typical Evolution Profile of CE for a Random Network in the Neuro-fuzzy Mixed-weight Algorithm

B.3 Graphical Display for CE Slope Estimation Buffer Size Variations

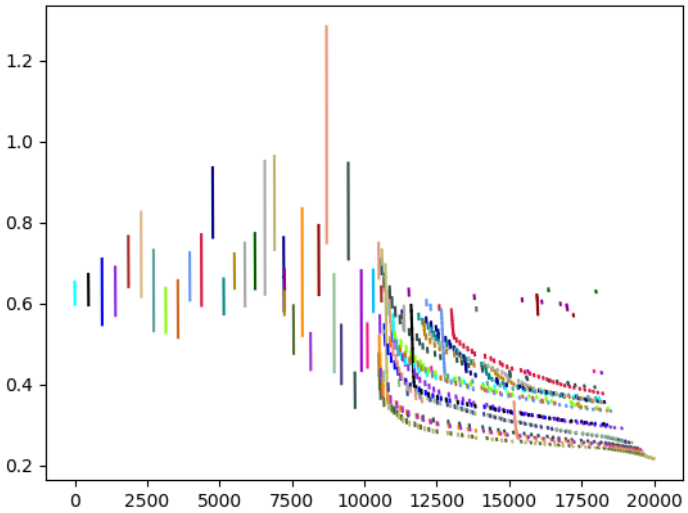


FIGURE B.3 – Buffer Size = 1*NBREP

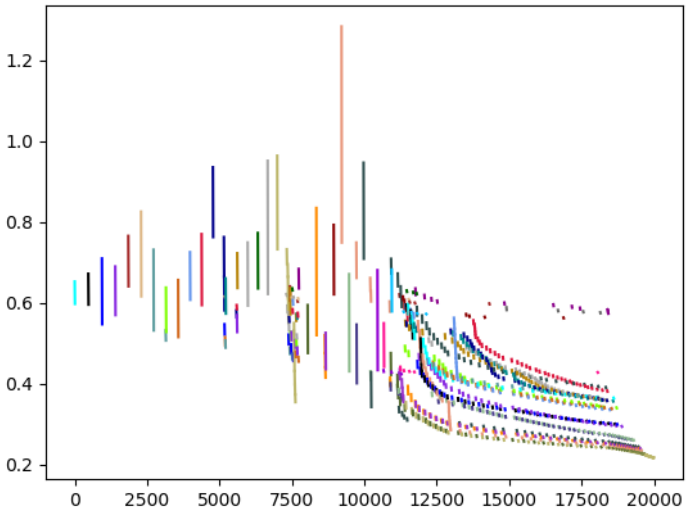


FIGURE B.4 – Buffer Size = 5*NBREP

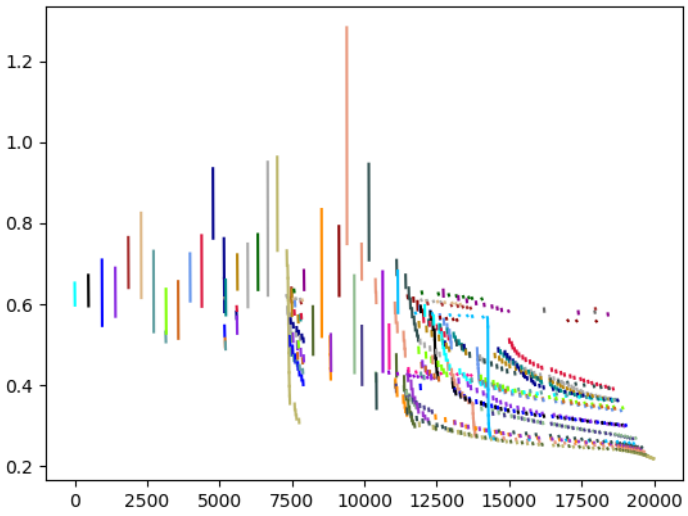


FIGURE B.5 – Buffer Size = 10*NBREP

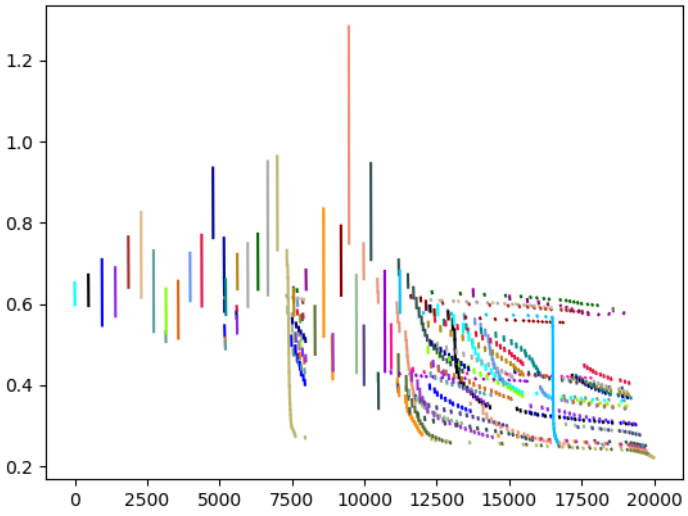


FIGURE B.6 – Buffer Size = 20*NBREP

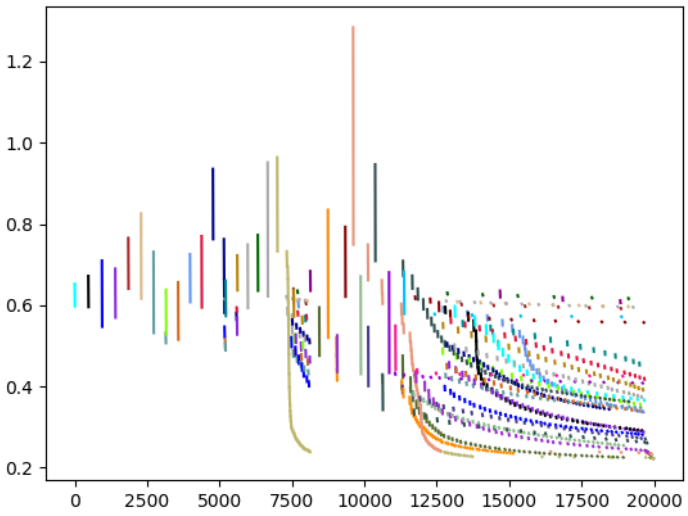


FIGURE B.7 – Buffer Size = 50*NBREP