Rapport de stage :
Réseaux neuroflous à poids mixtes

Thomas Sesmat
Supervisé par Jean-Loup Farges , Gauthier Picard et Filipo Perotto

15 novembre 2023

Remerciements

Ces remerciements ne sont pas classés par ordre d'importance.

Je remercie très sincèrement M. Phillipe Villedieu, Mme Béatrice Laurent Bonneau ainsi que le service des stages de l'INSA et le département GMM pour leur disponibilité et surtout très grande réactivité au début de ce stage qui a commencé bien difficilement.

Je remercie très chaleureusement M. Jean-Loup Farges pour l'immense aide apportée a tout instant, sa très grande pédagogie et sa patience envers moi.

Je remercie aussi M. Filipo Perotto pour son aide apportée et son souci du bien être de son co-bureau temporaire.

Je remercie Gauthier Picard pour son aide et sa bienveillance.

Je remercie l'ensemble de l'administration de l'ONERA et le service du DTIS pour son accueil et mon intégration.

Je remercie enfin mes parents pour leur disponibilité et tout leur soutien.

Je remercie à nouveau mes trois encadrants, M. Jean-Loup Farges, M. Filipo Perotto et M. Gauthier Picard pour leur aide, leur bienveillance et leur confiance en moi, me permettant de me sentir vraiment force proposition tout au long de ce stage.

:)

Résumé

De plus en plus d'études alarment sur l'importance de l'interprétabilité en intelligence artificielle. La possibilité de biais de raisonnement et de discrimination restreignent l'utilisation de l'apprentissage machine dans les domaines critiques tel que l'aviation ou la santé où les modèles appris se doivent d'être transparents.

En dépit du nombre croissant de groupes de recherche sur le sujet, aucune étude jusqu'alors n'est parvenue a concevoir un algorithme d'inférence permettant une transparence satisfaisante sur ses prises de décisions.

Ce stage a porté sur l'étude d'une piste prometteuse : les réseaux neuroflous à poids mixtes. Ce sont des algorithmes d'inférence combinant des réseaux de neurones et la logique floue. L'intégration de la logique floue et la conception de règles adaptables impliquent l'utilisation de poids booléens sur certaines couches du réseau, tandis que les autres paramètres de ce réseau sont des variables continues. L'apprentissage de ce réseau hybride se base donc sur une optimisation à variables mixtes.

Ce stage a permis la conception d'un algorithme d'apprentissage intégrant une recherche Tabou adaptée et une rétropropagation classique. De plus, des réflexions sont présentées et étudiées sur une méthode de traduction des différents poids du réseau après apprentissage. Les résultats obtenus au terme de ce stage permettent de répondre partiellement aux nombreuses questions de cette problématique et offrent des perspectives prometteuses. Cette voie reste en cours de développement, afin de résoudre les défis identifiés jusqu'alors.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Contexte

Ce rapport porte sur un stage de Master 1 effectué à l'ONERA (Office National d'Etudes et de Recherches Aérospatiales) de juillet à octobre 2023. Dans ce rapport, je décris formellement mes travaux sur le développement d'un réseau neuroflou à poids mixtes. L'environnent, l'état de l'art et le sujet du stage sont présentés dans ce chapitre, puis les améliorations apportées au réseau initial sont décrites dans le chapitre 2. Ensuite, l'algorithme d'apprentissage développé pour ce réseau est expliqué dans le chapitre 3. Les différents résultats obtenus et l'étude de l'influence des paramètres globaux sont détaillés dans le chapitre 5. Enfin, le chapitre 6 est réservé à la conclusion sur le sujet et le stage.

1.1 Aperçu de l'environnement

1.1.1 L'ONERA – *The French Aerospace Lab*

Créé en 1946, l'ONERA (Office National d'Etudes et de Recherches Aérospatiale est le centre français de recherche aérospatiale. Ses missions sont les suivantes

- Développer et diriger les recherches dans le domaine aérospatial ;
- Concevoir, mettre en oeuvre et exécuter la conduite de ces recherches ;
- Assurer la diffusion des résultats de ces recherches aux niveaux national et international, en coordination avec les services ou organismes chargés de la recherche scientifique et technique ;
- Promouvoir leur valorisation par l'industrie aérospatiale ;
- Faciliter leur application potentielle en dehors du domaine aérospatial.

L'ONERA compte environ 2000 employés, dont 1500 chercheurs, ingénieurs et techniciens répartis sur huit sites en France, présentés sur la Figure 1.1. Pour relever les défis importants des industries de l'aérospatiale et de la défense, il s'est doté d'installations expérimentales uniques en Europe. Tous les grands programmes aérospatiaux civils et militaires en France et en Europe portent une partie de l'ADN de l'ONERA : Ariane, Airbus, Falcon, Rafale, missiles, hélicoptères, moteurs, radars, etc.

1.1.2 Département scientifique et unité de recherche

Les recherches menées à l'ONERA sont structurées en 4 branches correspondant à des domaines disciplinaires majeurs : Mécanique des fluides et énergétique, Physique, Matériaux et structures, Traitement de l'information et systèmes. Au sein de chaque branche, plusieurs départements scientifiques sont déployés, et contribuent à la diversité des activités de recherche. Au centre ONERA de Toulouse, nous pouvons citer les départements suivants avec leurs domaines de recherches spécifiques :

- DEMR : Electro-Magnétisme et Radar

FIGURE 1.1 – Centres ONERA en France (2018)

— DMPE : Multi-Physique pour l'Energie
— DOTA : Optique et Techniques Associées
— DPHY : PHYsique, instrumentation, environnement et espace
— DTIS : Traitement de l'Information et Systèmes

DTIS

Le Département Traitement de l'Information et Systèmes (DTIS), où j'ai effectué mon stage, mène des études et des recherches pour maîtriser la conception, l'exploitation et l'autonomie des systèmes aérospatiaux. Le département applique son expertise dans les domaines de l'aéronautique, de l'espace et de la défense, avec les principales applications suivantes :

— Aéronefs (avions de transport, avions de chasse, drones, dirigeables, *etc.*)
— Systèmes aérospatiaux (systèmes de transport aérien, lanceurs, satellites, *etc.*)
— Systèmes d'information (systèmes de surveillance, systèmes de localisation, *etc.*)
— Systèmes de défense (missiles, systèmes de systèmes, *etc.*)

SYD

Au sein du DTIS, je faisais partie de l'unité SYstèmes intelligents et Décision (SYD) qui se concentre sur les méthodes formelles discrètes de résolution de problèmes, les systèmes multi-agents et la modélisation des besoins. Ainsi, les principaux domaines étudiés dans l'unité sont la Recherche Opérationnelle, l'Intelligence Artificielle (IA), l'ingénierie des exigences, la planification, les algorithmes et la combinatoire, et l'allocation et la prise de décision collaborative. Mes encadrants ici pendant ce stage de 12 semaines, sont Jean-Loup FARGES, Filipo PEROTO et Gauthier PICARD.

1.2 Etat de l'art

Depuis ses débuts dans les années 1950, avec les travaux de Warren McCulloch et Walter Pitts sur la cybernétique et de John Von Neumann et Alan Turing avec la conception de calculateurs en logique binaire, les travaux en IA sont traditionnellement classés en deux approches : ceux qui relèvent de l'IA symbolique et ceux qui relèvent de l'IA connexionniste [1], [2].

L'IA symbolique cherche à reproduire de manière formelle le raisonnement humain [3]. Elle utilise des règles et des symboles fixes essentiellement booléens pour définir une succession logique de

raisonnement. Elle fut notamment utilisée dans les algorithmes de traduction, où le passage d'une langue à une autre se fait par une suite de règles bien définies (bien que l'IA connexionniste, l'ait supplantée sur ce domaine de nos jours). Par l'introduction de règles expertes compréhensibles par l'Homme (si "évènement" alors "conséquence"), l'IA symbolique possède, grâce à la transparence des systèmes qu'elle produit, une grande interprétabilité et il est aisé de comprendre son fonctionnement. Très présente dans les années 1970 à 1990, elle fut cependant mise à l'écart à cause de sa rigidité dans sa prise de décision et de la difficulté à gérer des situations de plus en plus complexes, chaque règle devant être définie par l'humain.

L'IA connexionniste, soit les Réseaux de Neurones (RN), cherchent à reproduire le fonctionnement du cerveau humain. Les RN ont la capacité d'"apprendre" les relations mathématiques entre une série de variables d'entrée et de sortie correspondantes. Les réseaux sont entraînés afin d'ajuster leurs poids internes en fonction des relations mathématiques identifiées entre les entrées et les sorties sur un ensemble de données connues [4]. Ils gagnèrent en popularité dans les années 1980 avec l'invention de l'algorithme de rétropropagation des erreurs [5]. Depuis, de nombreux réseaux différents furent implémentés et les RN sont très largement utilisés dans de nombreux domaines tel que la reconnaissance d'image, le traitement du signal ou encore la simulation. Ils permettent une grande précision, mais étant plus complexes, les RN sont aussi perçus comme des "boites noires". Il est difficile de justifier et d'interpréter la pertinence de la configuration d'un réseau autrement que par un gain de performance, comme par exemple un gain de précision.

Ce manque d'interprétabilité pose des limites concrètes à l'utilisation des RN dans les domaines critiques tel que l'aéronautique, la médecine ou la finance. De plus, avec les scandales liés aux biais d'IA (préjugés sexistes, racistes dûs aux données d'apprentissages ou à la conception de l'algorithme) et l'identification de biais notables lors de l'apprentissage [6, 7], le courant actuel mondial cherche à développer des programmes implémentant certaines règles étiques. Il est donc indispensable de pouvoir justifier le fonctionnement d'un programme. De nombreux groupes de recherche ont d'ailleurs été mis en place ces dernières années afin de répondre à ces problématiques [8, 9, 10]. L'AI Act, une législation au niveau européen, est également en développement depuis quelques années afin de réguler l'utilisation de l'intelligence artificielle [11].

1.2.1 Logique floue

La logique floue est une théorie des ensembles introduite par Lotfi A. Zadeh en 1965 [12], qui étend la logique binaire d'appartenance en prenant en compte les degrés de vérité. Elle permet une représentation nuancée des données incertaines et des processus décisionnels complexes. Elle est particulièrement intéressante pour retranscrire la logique de décision humaine. La logique floue repose sur quatre éléments fondamentaux : les ensembles flous, les fonctions d'appartenance, les opérations de logique floue et l'inférence floue.

Ensemble flou

Dans la théorie classique des ensembles, un élément appartient à un ensemble (avec une valeur d'appartenance de 1) ou n'y appartient pas (avec une valeur d'appartenance de 0). En revanche, pour un ensemble support X, un ensemble flou A est défini sur un univers de discours $\mathbb{D}$ comme l'application d'une fonction de X à l'intervalle [0, 1]. Chaque élément x dans X se voit attribuer un degré d'appartenance, noté $\mu_A(x)$, qui représente dans quelle mesure x appartient à A. Formellement, un ensemble flou A peut être représenté comme suit :

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

	T-norme de Gödel	T-norme produit
ET Flou	$\mu_C(x) = \min(\mu_A(x), \mu_B(x))$	$\mu_C(x) = \mu_A(x) \cdot \mu_B(x)$
OU Flou	$\mu_C(x) = \max(\mu_A(x), \mu_B(x))$	$\mu_C(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$
NON Flou	$\mu_C(x) = 1 - \mu_A(x)$	

TABLE 1.1 – Opérateurs logiques pour la T-norme de Gödel et la T-norme produit. "$\cdot$" correspond au produit numérique réel

Fonction d'appartenance

Une fonction d'appartenance $\mu_A(x)$ quantifie le degré d'appartenance d'un élément x à un ensemble flou A. Elle caractérise à quel point x appartient à l'ensemble de manière graduelle. Lorsque X correspond à un ensemble de vecteurs de variables réelles, les fonctions d'appartenance peuvent adopter différentes formes mathématiques, notamment triangulaire, gaussienne, ou encore sigmoïdale. Le choix de ces fonctions dépend de la nature du problème à modéliser et de la façon dont l'incertitude doit être représentée.

Opérations Logiques Floues

Les opérations de logiques floues étendent les opérations booléennes afin de les adapter aux degrés d'appartenance. Il existe plusieurs définitions des opérateurs logiques selon la norme triangulaire (T-norme) utilisée, le tableau 1.1 présente les expressions des opérateurs pour les deux T-normes les plus utilisées : la T-norme de Gödel (ou norme minimale) et la T-norme produit :

— ET ($\wedge$) Flou : Pour deux ensembles flous A et B avec des fonctions d'appartenance $\mu_A(x)$ et $\mu_B(x)$, l'opération **ET flou** produit un nouvel ensemble flou C qui a pour fonction d'appartenance

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

, où T est la T-norme considérée.

— OU ($\vee$) Flou : Pour les ensembles flous A et B, l'opération **OU flou** produit un nouvel ensemble flou C avec la fonction d'appartenance $\mu_C(x)$ définie comme suit :

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

, où $T*$ et la T-conorme associée à T.

— NON ($\neg$) Flou : Pour un ensemble flou A avec une fonction d'appartenance $\mu_A(x)$, l'opération **NON flou** la négativise pour créer un nouvel ensemble flou C avec la fonction d'appartenance $\mu_C(x)$ définie comme suit :

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

Il s'agit de l'opérateur de négation canonique, dont l'expression est commune à toutes les T-normes.

La T-norme produit est différentiable et est basée sur l'assimilation de l'appartenance à une fréquence, la représentativité des fréquences et l'indépendance des appartenances. La T-norme de Gödel est la seule T-norme présentant des propriétés d'idempotence, d'absorption et de distributivité [13]. Par contre elle n'est pas différentiable lorsque $\mu_A(x) = \mu_B(x)$.

Inférence floue

L'inférence floue est un processus qui utilise les règles de logique floue et les données d'entrée pour déduire une sortie nette. Elle implique les étapes suivantes :

— Fuzzyfication : Conversion des données d'entrée nettes en ensembles flous en utilisant des fonctions d'appartenance appropriées.
— Évaluation des Règles : Application des règles de logique floue qui relient les ensembles flous d'entrée aux ensembles flous de sortie via les opérateurs de logique floue.
— Agrégation : Combinaison des ensembles flous de sortie pour obtenir un ensemble flou composite qui correspond à l'ensemble des règles établies pour toutes les sorties possibles.
— Défuzzyfication : Conversion de l'ensemble flou composite en une valeur nette. L'objectif est de transformer les résultats des couches précédentes en donnée numérique. D'après JSR. Jang, il en existe trois types [14] : la moyenne pondérée des sortie nette pour chaque règle induite par les couches précédentes, le maximum des sorties nettes des couches précédentes, ou l'utilisation des règle *if-then* de Takagi-Sugeno [15].

1.2.2 Rétropropagation et descente de gradient

La rétropropagation est une méthode qui permet de calculer efficacement les gradients pour chaque couche composée de variables numériques d'un réseau de neurones. Elle fonctionne de manière récursive, en propageant l'erreur du réseau de la sortie vers l'entrée. L'algorithme 1 présente en pseudo code les étapes clés d'une rétropropagation du gradient, en y intégrant la descente de gradient.

Algorithme 1 : Rétropropagation du gradient

```
Input : data - données d'apprentissage du réseau, correspondant aux scores d'entrées et les
        vecteur de sorti correspondant
Input : pas - taux d'apprentissage de la descente de gradient
Input : var - vecteur des variables numériques du réseau
Output : var_app - variables numériques du réseau entraînées
grad_cum ← 0;
for (x, y) ∈ data do
    y_pred ← ForwardPropagation(x);
    grad_cum ← BackwardPropagation(y_pred, y);
end
grad ← grad_cum/data.taille ;
var_app ← GradDescent(var, pas, grad);
```

Lors de la première étape, le réseau effectue, pour chaque couple de vecteurs entrée-sortie, par la fonction *ForwardPropagation()* la transformation de l'entrée en une sortie prédite selon les différentes couches du réseau. Une fois la prédiction obtenue, la fonction *BackwardPropagation()* permet, à partir de la couche de sortie, de calculer les gradients de l'erreur relative à chaque paramètre numérique (*i.e.* poids et biais numériques) du modèle en utilisant la règle de la chaîne. La valeur obtenue est ensuite stockée dans *grad_cum*. Une fois l'ensemble des données parcourus, le gradient moyen *grad* est obtenu en divisant *grad_cum* par le nombre de couple de données. Enfin, la fonction *GradDescent()* effectue une descente de gradient, afin d'ajuster les paramètres numérique du modèle, définie comme :

$$var_{app} = var - pas * grad$$

Lors de l'apprentissage sur un réseau de neurones, l'algorithme 1 est répété pour un certain nombre d'itérations (époques) ou jusqu'à ce qu'un critère d'arrêt soit atteint, par exemple, lorsque la fonction de coût atteint un seuil acceptable ou que la variation entre deux itérations successives est en deçà d'une certaine valeur.

1.2.3 Méthode tabou

Nous présentons ici une technique qui n'est pas exactement utilisée comme telle dans notre réseau, mais dont nous nous inspirons. L'algorithme Tabou est une méthode d'optimisation et de recherche heuristique qui vise à trouver la meilleure solution possible dans un espace de recherche complexe tout en évitant de rester bloqué à des optima locaux. Une description des étapes clés de la méthode est présentée dans l'algorithme 2.

Algorithme 2 : Méthode Tabou en optimisation

Input : s_ini - la solution initiale
Input : anciennete - critère d'ancienneté de la liste taboue
Input : crit_arrêt - Critère d'arrêt (ex : nombre d'itération, différence minimale entre deux solutions considérées)
Output : s_best - la meilleur solution trouvée une fois le critère d'arrêt atteint

```
s_best ← s_ini;
tabou ← None;
while notcrit_arrêt do
    voisin_s ← GenerVoisin(s_best);
    best_voisin ← SelecBestNonTabou(voisin_s);
    tabou ← UpdateTabou(tabou, best_voisin, anciennete);
    if best_voisin < s_best then
        s_best ← best_voisin;
    end
    crit_arrêt ← UpdateCrit(crit_arrêt);
end
```

A partir d'une solution s_best considérée idéale en l'état, $GenerVoisin()$ fait une recherche locale autour de la solution actuelle pour identifier les solutions voisines $voisin\_s$. Puis, parmi tous les voisins, $SelectBestNonTabou()$ sélectionne la meilleure solution non taboue, c'est à dire qui n'a pas été récemment explorée selon le critère d'*anciennete*. Alors $UpdateTabou$ met à jour la liste *tabou* en ajoutant la solution récemment explorée et retirant celles qui ne sont plus taboues selon ce même critère d'*anciennete*. La solution idéale en l'état est remplacée par son meilleur voisin si ce dernier conduit à un meilleur résultat. Sinon, on reste sur cette solution et recherche un nouveau voisin avec la liste *tabou* actualisée. En fin de boucle, le critère d'arrêt est actualisé par la fonction $UpdateCrit()$.

1.2.4 Réseaux neuro-flous (RNF)

Plusieurs pistes ont été ouvertes pour faire avancer l'XAI (Explanable AI) [16], [17]. Une méthode possible sont les réseaux neuro-flous [14]. Ce sont des réseaux hybrides à mi-chemin entre l'IA symbolique et connexionniste, via l'introduction de la logique floue de Zadeh dans les réseaux de neurones. L'intégration de cette logique dans les réseaux de neurones permet de profiter de ses deux avantages : la gestion du manque de précision, mais aussi la modélisation d'un raisonnement proche du cerveau humain, et donc plus interprétable. Les RNF manipulent des descripteurs linguistiques plutôt que des entrées numériques, et les règles floues décrivent alors les relations entres ces différents descripteurs. Le réseau apprend à travers des exemples et des corrections d'erreurs, comme dans les réseaux de neurones traditionnels, mais utilise des fonctions d'appartenances floues pour décrire les entrées. Leur structure s'inspire des schémas d'inférence floue : les couches successives du réseau serfant à réaliser fuzzyfication, inférence et défuzzification comme décrit précédemment. Une analyse récente de la

littérature sur le sujet conclut que les modèles de réseaux neuro-flous et leurs dérivés sont efficaces pour construire des systèmes présentant un degré élevé de précision et pouvant être interprétés à un niveau approprié, ceci pour de nombreux domaines de l'économie et des sciences [18].

Cependant très peu d'études jusqu'alors se concentrent sur la sélection dynamique des règles d'apprentissage du réseau afin d'améliorer la règle experte initiale. On peut tout de même citer l'étude de G. Marra et al. [19], qui présente un réseau de type DCR (*Deep Concept Reasoner*) capable de sélectionner des règles à partir d'un plongement (*embedding* en anglais) et qui fournit d'excellents résultats aussi bien sur l'interprétation de ses règles que sur les performances de classification. Cependant l'utilisation d'*embedding* apporte deux problèmes : il est nécessaire de faire un travail non trivial sur les données (quelque soit leur type) avant d'utiliser le réseau DCR, et les résultats issus ne sont pas forcement interprétables par l'homme dû à l'*embedding*. Ainsi il y a un risque que le problème ne soit que "déplacé" et non résolu.

L'objectif de ce stage est donc de contribuer à un projet visant l'amélioration et l'apprentissage de règles en étudiant sur un réseau simple le concept d'amélioration dynamique de la règle experte par l'optimisation des poids booléens permettant de configurer cette règle en plus des poids et biais numériques. Nous souhaitons également effectuer le moins possible de transformations sur les données en amont du réseau, afin de pouvoir conserver leur interprétabilité et diminuer au maximum le temps de calcul global. Le projet initial vise à aborder les défis scientifiques suivants :

- La définition précise de fonctions pour certains noeuds du réseau et la caractérisation des dérivées nécessaires à la rétropropagation pour ces fonctions.
- Le traitement du problème d'apprentissage comme un problème à variables mixtes.
- La définition de règles de simplification afin d'obtenir des règles cohérentes et interprétables par l'homme.

Dans cette optique, nous souhaitons également maîtriser totalement l'implémentation en utilisant le moins possible les librairies de fonctions préconstruites (type *Scikit learn*, *Pytorch*, *etc*), dans la mesure du possible.

1.2.5 État du projet en début de stage

Ce stage fait suite à un projet long de l'ENSEIHT (équivalent à un projet de recherche de 4A à l'INSA). Lors de ce projet long, les étudiants avaient déjà codé une partie de la structure réseau. Nous allons dans la suite faire l'état du projet au début de ce stage. En dépit des nombreuses remarques faites dans cette partie je ne cherche en aucun cas à réduire le travail des étudiants ayant effectué ce projet long mais simplement faire un retour objectif sur l'état du projet lorsque je l'ai récupéré pour le stage afin de pouvoir présenter clairement les différentes avancées faites durant ce dernier. Je tiens d'ailleurs à préciser que c'est en partie grâce à leur début de code ainsi qu'à leur rapport que j'ai pu être rapidement efficace pour le stage.

Réseau à poids mixtes

La structure initiale du réseau pour la partie *feedforward* fut implémentée, présentée figure 1.3. Il s'agit d'un perceptron multicouche contenant quatre couches cachées : Floutage, Conjonction, Disjonction et Normalisation. Pour chaque score, deux descripteurs leur sont associés, un premier représentant le cas "le score est grand" et l'autre "le score est petit", dont les fonctions d'appartenance sont des sigmoïdes. Une visualisation des deux fonctions d'appartenances type est représentée figure 1.2. La norme triangulaire utilisée est celle de Gödel.

Couche de floutage :

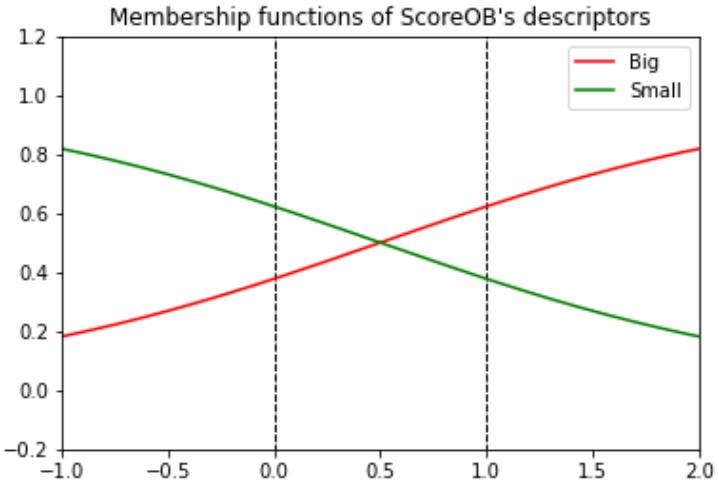


FIGURE 1.2 – Exemple : visualisation des fonctions d'appartenance du score *score boite orange* (jeu de données crash d'avion))

Cette première couche permet d'intégrer la logique floue au réseau et permet l'association des descripteurs linguistiques aux différents scores. Chaque descripteur linguistique est caractérisé par une sigmoïde de la forme :

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

où w et b sont respectivement le poids et le biais associés à cette sigmoïde. Dans notre cas, on considère deux descripteurs linguistiques pour chaque score, correspondant à *le score est élevé* et *le score est faible*.

On décompose la couche de la façon suivante : elle est constituée de blocs de sigmoïdes dont le nombre correspond au nombre de scores en entrée, noté ici n_{in}. Puis chaque bloc i est constitué de de $n_{desc,i}$ sigmoïdes, correspondant au nombre de descripteurs associés à ce bloc. Pour résumer, la couche de floutage va donc renvoyer $n_{out}^{fuz} = \sum_{i=1}^{n_{in}} n_i^{desc}$ valeurs où la j^{eme} valeur du bloc i vaudra :

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

L'implémentation algorithmique est effectuée sous forme de classe. On implémente trois classes imbriquées :

- *SingleInputSigmoid* qui est initialisée à partir d'un poids et d'un biais et qui créée une cellule contenant les informations (poids et biais) du descripteur considéré pour un score donné. Cette classe permet notamment de calculer via la méthode *activate*() la valeur d'appartenance d'un score par rapport à ce descripteur.
- *SingleInputMultiSigmoidBlock* qui est initialisée par une liste de poids et de biais qui créée une liste de cellules (un bloc) de type *SingleInputSigmoid*. Cette classe permet notamment de calculer via la méthode *activate*() faisant appel à la classe *SingleInputSigmoid* les valeurs d'appartenance d'un score donné pour les descripteurs qui lui sont associés.
- *FuzzificationLayer* qui est initialisée par une liste de listes de poids et de biais et qui créée l'ensemble de la couche de floutage via une liste de bloc de type *SingleInputMultiSigmoidBlock*. Cette classe permet notamment de calculer via la méthode *activate*(), faisant appel à la classe *SingleInputMultiSigmoidBlock*, pour chaque score l'ensemble de ses valeurs d'appartenance pour les descripteurs qui lui sont associés.

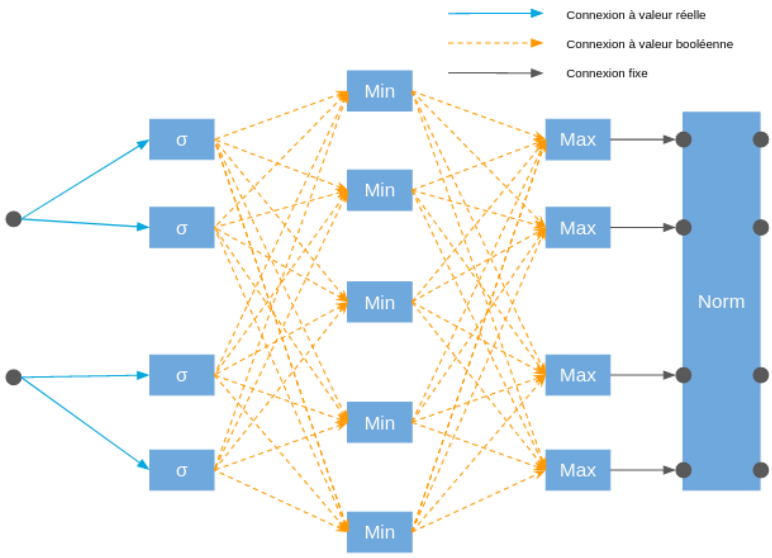


FIGURE 1.3 – Structure typique d'un réseau neuroflou tel qu'implémenté

Couche de conjonction

Cette seconde couche permet la conjonction de valeurs d'appartenances obtenues après la première couche par utilisation d'un pseudo-minimum sur $[0;1]$ défini comme suit :

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

où w_j^{bool} est une liste de poids booléens réglant la prise en compte ou non des différentes valeurs x_j contenues dans le vecteur x pour le calcul du minimum. Ce pseudo-minimum correspond à la valeur logique **ET** pour la logique floue de Zadeh qui utilise la T-norme de Gödel.

On décompose la couche de la façon suivante : elle est constituée de cellules logiques de conjonction dont le nombre est défini de manière arbitraire, noté ici n_{conj}. Chaque poids associé à une cellule correspond à un vecteur de n_{out}^{fuzz} valeurs booléennes. Pour résumer, la couche de conjonction va donc renvoyer $n_{out}^{conj} = n_{conj}$ valeurs numériques où la valeur en sortie de la cellule i vaudra :

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

On implémente deux classes imbriquées pour définir cette couche :

— *AndCell* qui est initialisée à partir d'une liste de poids booléens et qui créée une cellule contenant les informations (poids). Cette classe permet notamment de calculer via la méthode *activate*() la valeur du pseudo-minimum comme défini dans (1.3) à partir de la sortie de la couche précédente.

— *AndLayer* qui est initialisée par une liste de listes de poids booléens et qui créée une liste de cellules de type *AndCell*. Elle correspond à l'ensemble de la couche de conjonction. Cette classe permet notamment de calculer via la méthode *activate*() faisant appel à la classe *AndCell* les valeurs de pseudo-minimum pour l'ensemble des cellules qui composent la couche.

Couche de disjonction

Cette troisième couche permet la disjonction des valeurs obtenues à la couche précédente par l'utilisation d'un pseudo-maximum sur $[0;1]$ défini de manière analogue au pseudo-minimum :

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

où w_j^{disj} est une liste de poids booléens réglant la prise en compte ou non des différentes valeurs x_j contenues dans le vecteur x pour le calcul du maximum. Ce pseudo-maximum correspond à la valeur logique **OU** pour la logique floue de Zadeh.

Nous décomposons la couche de la façon suivante : elle est constituée de cellules logiques de disjonction dont le nombre est défini comme étant le nombre de classes de notre problème, noté ici n_{disj}. En effet, dans un perceptron plus développé le nombre de cellules pourrait être variable, mais ici, il n'y a que la couche de normalisation qui vient ensuite. Or la couche de normalisation ne fait pas varier le nombre de sorties par rapport à ses entrées. C'est pourquoi il est nécessaire dès à présent d'avoir le bon nombre de sorties par rapport au problème. Chaque poids associé à une cellule correspond à un vecteur de n_{out}^{conj} valeurs booléennes. Pour résumer, la couche de conjonction va donc renvoyer $n_{out}^{disj} = n_{disj}$ valeurs numériques où la valeur en sortie de la cellule i vaudra :

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

On l'implémente de façon strictement analogue à la couche de conjonction, en utilisant simplement la fonction maximum au lieu de minimum.

Couche de normalisation

Cette dernière couche permet de normaliser les résultats obtenus à la couche précédente afin d'obtenir pour chaque vecteur de scores considéré initialement de dimension n_{in} une probabilité d'appartenir à chacune des classes. La couche de normalisation va retourner un vecteur de taille n_{out}^{norm} qui est égale au nombre de classes de notre problème. La i^{eme} valeur du vecteur de sortie vaut :

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

où x_j et le j^{eme} scalaire associé à $x = (x_1, ..., x_n)$

Nous implémentons dans une seule classe cette couche. La classe $NormalizationLayer$ qui est initialisée à partir du nombre de classes qui composent notre problème. Cette classe permet notamment de calculer via la méthode $activate()$ la valeur normalisée de chacune des sorties de la couche précédente.

Calcul analytique des sensibilités

Afin de pouvoir améliorer notre réseau, nous allons utiliser la rétropropagation. La rétropropagation consiste à propager l'erreur de sortie du réseau à travers les différentes couches et à adapter les variables numériques en fonction. Afin d'obtenir de manière exacte la dérivée de la fonction de rétropropagation, nous allons définir les sensibilités de la sortie de chacune des couches par rapport à son entrée. La règle de la chaîne nous permettra ensuite de calculer la contribution de chaque neurones à l'erreur.

L'erreur choisir est l'entropie croisée, notée CE dans la suite et définie comme suit :

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

, où y_i est la prédiction souhaitée, et x_i le résultat obtenu après la propagation avant.

Cette fonction est bien adaptée pour traduire l'erreur entre deux distributions de probabilité car sa minimisation correspond au logarithme du maximum de vraisemblance des observations y_i pour la distribution modélisée par le réseau. Pour d'autres problèmes, un critère généralement utilisé est l'erreur quadratique moyenne.

Afin d'obtenir une valeur qui représente l'ensemble de l'échantillon, on fait la moyenne de CE notre fonction coût sur l'ensemble des valeurs obtenues pour le jeu de données :

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

où $x^{expc}(k)$ est le *one-hot* vecteur correspondant à la sortie effective pour le vecteur k-ème d'entrée.

Pour plus de simplicité d'écriture, on ne se base que sur un seul échantillon dans la suite de cette partie.

Le critère

La sensibilité du critère par rapport à x_i^{norm}, un paramètre de la couche précédente (normalisation), vaut :

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

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

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

En utilisant la formule dérivation de la composée de deux fonctions on obtient alors :

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

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

où z est le paramètre du réseau par rapport auquel on souhaite dériver CE.

La couche de normalisation

On décompose la dérivée suivante

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

Avec $(i, j) \in [\![1; n^{norm}]\!] \times [\![1; n^{disj}]\!]$.

REMARQUE : En pratique, on a $n^{norm} = n^{disj}$ (cf 1.3)

On dérive l'expression de la couche de normalisation :

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

La couche de disjonction
Avec l'utilisation de la norme T de Gödel, la couche de disjonction correspond à l'utilisation d'un pseudo maximum, comme défini dans 1.2.1. La rétro propagation du gradient à travers cette couche consiste à propager le gradient uniquement à l'indice où la valeur maximale est obtenue lors de la propagation directe, et mettre à zéro le gradient des autres indices. Cela permet de ne propager le gradient qu'à la seule entrée qui a contribué à la sortie maximale, et donc de ne pas affecter les autres entrées.

On a alors, pour $(j,i) \in [\![1; n^{disj}]\!] \times [\![1; n^{conj}]\!]$:

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

avec

$$\frac{\partial x_j^{disj}}{\partial x_i^{conj}} = \begin{cases} 1 & \text{si } x_i^{conj} * w_{i,j}^{disj} \text{ est la valeur donnant le max} \\ 0 & \text{sinon} \end{cases} \tag{1.15}$$

La couche conjonction
La couche de conjonction correspondant cette fois-ci à un pseudo-min (voir à nouveau 1.2.1), on trouve un résultat quasi analogue à la couche précédente, pour $(i,j,k) \in [\![1; n^{disj}]\!] \times [\![1; n^{conj}]\!] \times [\![1; n_j^{fuzz}]\!]$:

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

$$\frac{\partial x_i^{conj}}{\partial x_{j,k}^{fuzz}} = \begin{cases} 1 & \text{si } x_{j,k}^{fuzz} \text{ est la valeur donnant la sortie minimale} \\ 0 & \text{sinon} \end{cases} \tag{1.17}$$

La couche de fuzzification
On a deux cas, selon si nous dérivons par rapport aux poids ou par rapport aux biais. Dans les deux cas, il s'agit d'une matrice diagonale dont les termes de la diagonale valent, pour $(j,k) \in [\![1; n^{conj}]\!] \times [\![1; n_j^{fuzz}]\!]$:

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

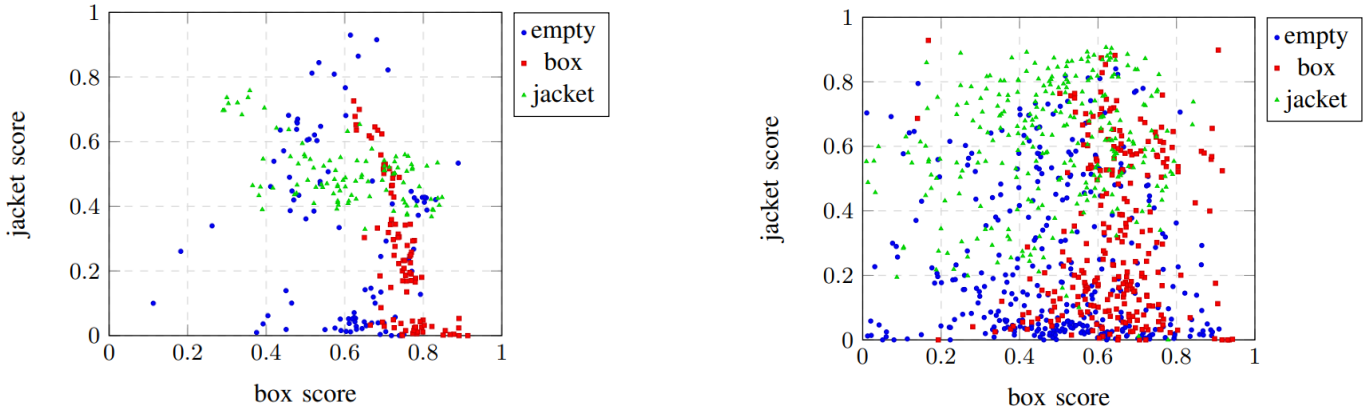
$$\frac{\partial x_{j,k}^{fuzz}}{\partial b_{j,k}} = \frac{1}{1 + e^{-(w_{j,k}x_j + b_{j,k})}} \left(1 - \frac{1}{1 + e^{-(w_{j,k}x_j + b_{j,k})}}\right) \tag{1.19}$$

Cas d'usage

Lors du projet long, l'ensemble du code à été implémenté sur un Jupyter Notebook, certains codes présentaient également des erreurs sur le fond mais aussi sur la forme (mauvaise optimisation du code). L'implémentation du calcul des sensibilités à été effectuée et une sensibilité numérique avait également été implémentée. Cependant, les calculs des sensibilités ont été implémentées à l'extérieur des structures de classe, ne tirant pas à profit leur intérêt. Enfin, les algorithmes de calculs de gradient (analytique et numériques) donnèrent des résultats différents, montrant la présence d'erreurs dans le code, ils n'ont donc pas été repris dans la suite.

Les données

Pour la construction de ce réseau, on se base initialement sur un problème de classification à partir d'un jeu de données modélisant un crash d'avion [20]. Ce jeu de données comprend trois classes : "Boite Orange", "Gilet Jaune", "Vide" et deux scores pour les décrire : "Score Veste Jaune" et "Score Boite Orange". Ces scores ont été acquis par un drone qui a pris différentes zones en photo, puis ces dernières passent par une suite d'instruction pour les transformer en scores. Une explication détaillée de l'ensemble du processus d'acquisition et de transformation des données en scores peut être trouvée dans l'article initial [20]. Les jeux de données de validation et d'apprentissage sont issus d'acquisitions effectuées à deux moments différents, séparés par plusieurs semaines d'intervalle.



(a) Données d'apprentissage

(b) Données de validation

FIGURE 1.4 – Représentation des classes en fonction des scores.

La figure 1.4 présente les différentes classes selon les deux scores pour les données de validation et les données d'apprentissage. Il y a 882 entrées pour les données de validation, chaque classe en proportion approximative de 1/3 et il y a 287 entrées pour les données d'apprentissage, chaque classe également en proportion 1/3. Les données sont séparées en six fichiers différents, ne contenant que les couples de scores, selon le type d'échantillon (validation/apprentissage) et de la classe dont il s'agit ("Boite Orange", "Gilet Jaune", "Vide").

1.3 Conclusion

Dans cette première partie, nous avons pu mettre en évidence un manque dans la littérature scientifique, et bien que d'autres recherches soient menées afin de répondre à la problématique, les résultats obtenus ne permettent pas de résoudre de façon convaincante le problème. D'autre part, les notions techniques et mathématiques du projet ont été clairement expliquées. Enfin, l'état du projet au début du stage fut exposé afin de pouvoir discerner les avancées effectuées au cours de ce dernier et celles effectuées auparavant par les étudiants de l'N7.

Chapitre 2

Les améliorations apportées au réseau et au cas d'usage

2.1 Modification des données

Le jeu de données initial ayant été acquis à deux moments différents, nous effectuons un test statistique de Student avec variances inégales afin de vérifier si il y a des différences significatives des moyennes entre les deux échantillons. Les moyennes et les écarts-types selon le score sur le jeu de données sont disponibles en annexe A.1 pour les deux échantillons considérés. Les p-valeurs issues de ces tests, présentées dans le Tableau 2.1a, nous montrent qu'il y a bien une différence significative entre les deux jeux de données.

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) Échantillons initiaux

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) Échantillons issus d'un échantillonnage stratifié

TABLE 2.1 – Test de Student entre les échantillons d'apprentissage et de validation sur les données du crash d'avion

Nous avons donc fusionné puis effectué un échantillonage stratifié afin d'obtenir des données plus homogènes (ratio 7/3 pour les échantillons apprentissage/validation). Un second test de Student, cf. Tableau 2.1b (moyennes et variances également présentes Annexe A.2), nous montre qu'il n'y a plus de différences significatives entre les moyennes des deux échantillons, et au vu de la méthode adoptée, nous pouvons conclure que ces deux échantillons sont valides pour le développement du réseau.

Le format des données a également été modifié afin d'avoir toutes les classes associées à un même échantillon dans le même fichier. Le format adopté est de la forme, pour une entrée :

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

où OB correspond à boite orange, YJ à gilet jaune et EB à vide.

Après plusieurs semaines sur ce jeu de données, nous avons préféré nous concentrer sur un jeu de données plus abordable, dont les classes sont moins enchevêtrées (voir figure 1.4). Le jeu de données sélectionné est "Breast Cancer Wisconsin (Diagnostic)". La méthode d'acquisition des données est disponible dans l'article initial [21]. Seule certaines variables ont été retenues. La version utilisée durant ce stage comporte 7 variables et 559 instances. Les différentes variables sont :

— Diagnostic : B : Bénigne, M : Maligne, Sortie - Type de cancer détecté. Ce sont les deux classes que l'on souhaite.
— Rayon (*Radius_m*), Entrée - moyenne des distances entre le centre et les points du périmètre
— Texture (*Texture_m*), Entrée - écart-type des valeurs en niveaux de gris
— Périmètre (*Perimeter_m*), Entrée - Périmètre de la tumeur
— Aire (*Area_m*), Entrée - Aire de la tumeur
— Concavité (*Concavity_m*), Entrée - Gravité des parties concaves du contour

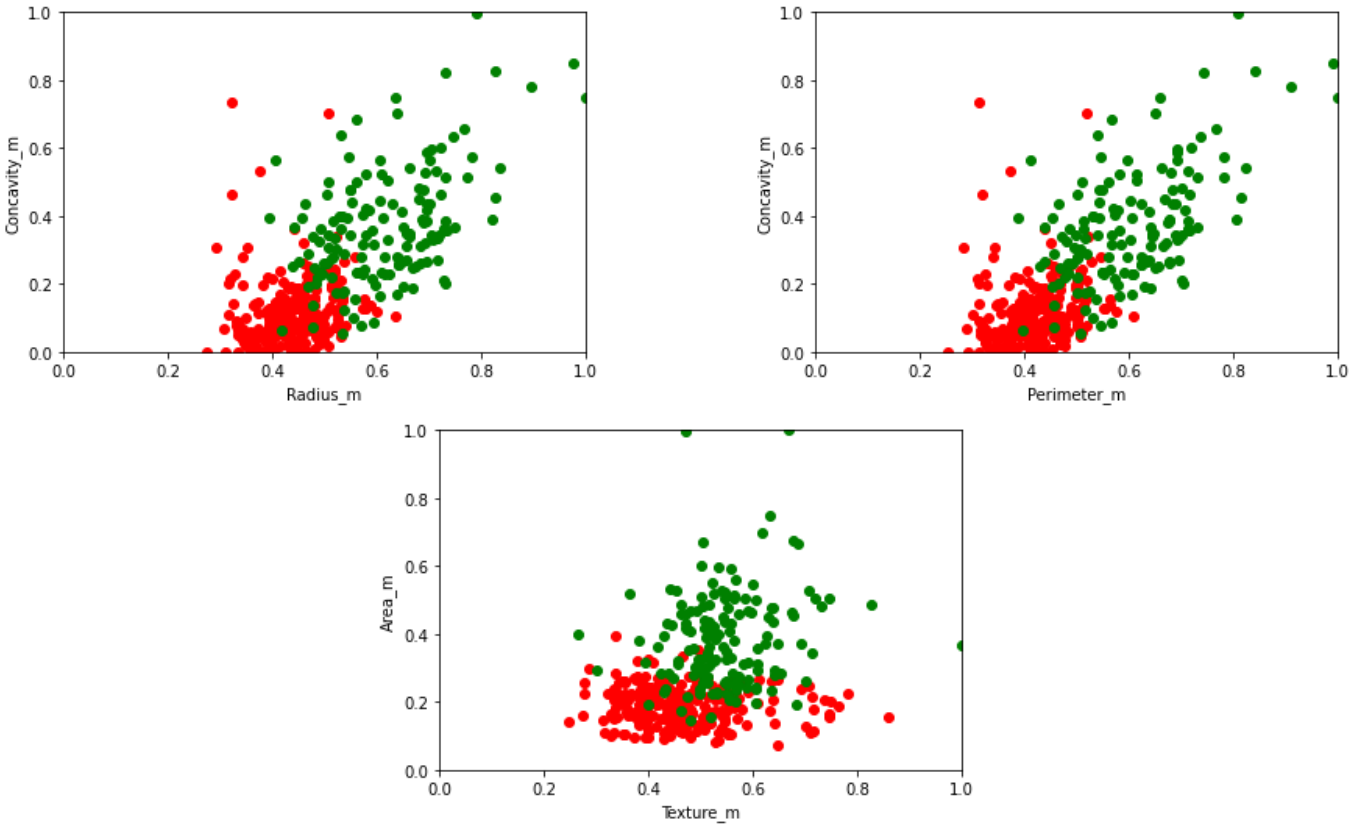


FIGURE 2.1 – Représentation des classes en fonction des scores pour le jeu de données Breast Cancer Wisconsin (Diagnostic). *Rouge* classe B ; *Vert* classe M

Les variables numériques ont été normalisées afin d'obtenir un score. Les données ont été formatées de façon analogue aux données précédentes. Un échantillonnage stratifié a également été effectué afin de séparer les données en un échantillon de validation et un échantillon d'entraînement (dans des proportion identique au jeu de données précédent). Le jeu de données étant assez petit, un test de Student a également été effectué après l'échantillonnage afin de s'assurer qu'il n'y ait pas de différences significatives entre les échantillons d'apprentissage et de validation. Les moyennes et écarts types sont disponibles en annexe A.3 Les résultats peuvent être observés sur la figure 2.2, qui nous permet de conclure que l'échantillonage stratifié nous a permis de faire deux échantillon corrects pour l'apprentissage.

La visualisation des classes en fonction de scores d'entrées sont également disponibles figure 2.1.

Classe	Score	P-valeur P
B	Radius_m	P = 0.9826
B	Texture_m	P = 0.8888
B	Perimiter_m	P = 0.2288
B	Area_m	P = 0.6295
B	Concavity_m	P = 0.8579
M	Radius_m	P = 0.4038
M	Texturem	P = 0.3942
M	Permiter_m	P = 0.3478
M	Area_m	P = 0.5417
M	Concavity_m	P = 0.4473

TABLE 2.2 – Test de Student entre les échantillons d'apprentissage et de validation sur les données Breast Cancer Wisconsin

On observe que pour chaque score, les classes peuvent être relativement bien séparées. On peut également observer certaines similitudes pour la distribution selon certaines variables. Les graphiques des corrélations, sur la figure 2.2 nous montrent que le rayon est très corrélé avec le périmètre ainsi qu'avec l'aire. Les tumeurs étant généralement de forme arrondie, on retrouve bien une corrélation linéaire entre le périmètre et le rayon, ainsi qu'une corrélation quadratique pour le rayon et l'aire.

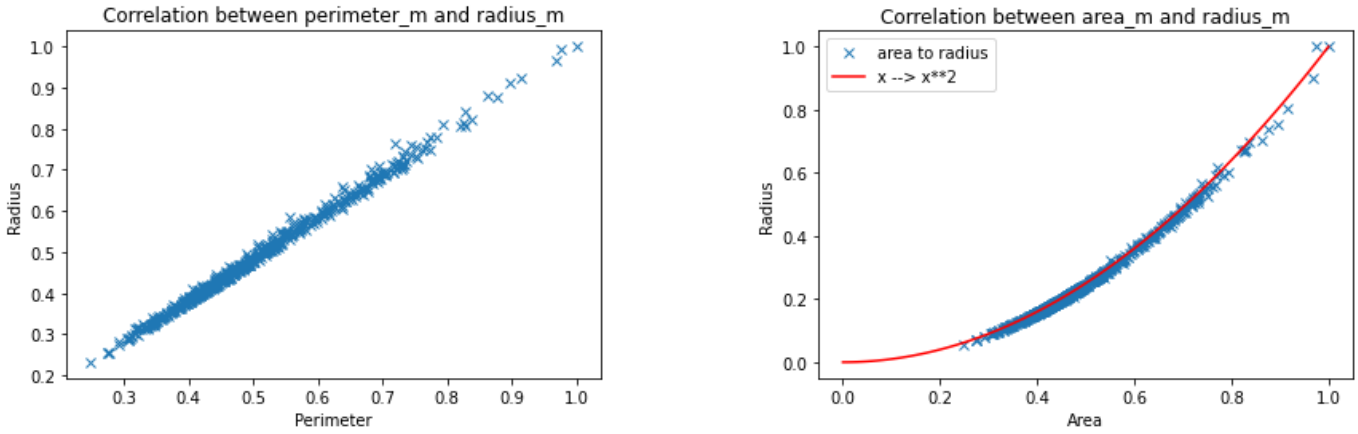


FIGURE 2.2 – Visualisation de la corrélation entre différents scores du jeu de données Breast Cancer Wisconsin

2.2 Modification théorique

Une vérification des calculs des sensibilités à été effectuée afin de s'assurer que l'erreur entre les gradients numérique et analytique développé lors du Projet Long n'était pas issue d'une erreur de calcul. Suite à ça, les formules de pseudo maximum et pseudo minimum ont été légèrement revues dans leur expression afin de correspondre mieux à la réalité algorithmique :

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 Vue d'ensemble de la structure algorithmique

2.3.1 Feedforward

Toutes les couches du réseau sont liées et initialisées dans la classe $NeuroFuzzyNetwork$ dont la méthode $activate()$ va permettre l'appel en cascade des différentes méthodes $activate()$ des différentes couches pour retourner la probabilité d'appartenance à chacune des classes pour un vecteur de scores donnés.

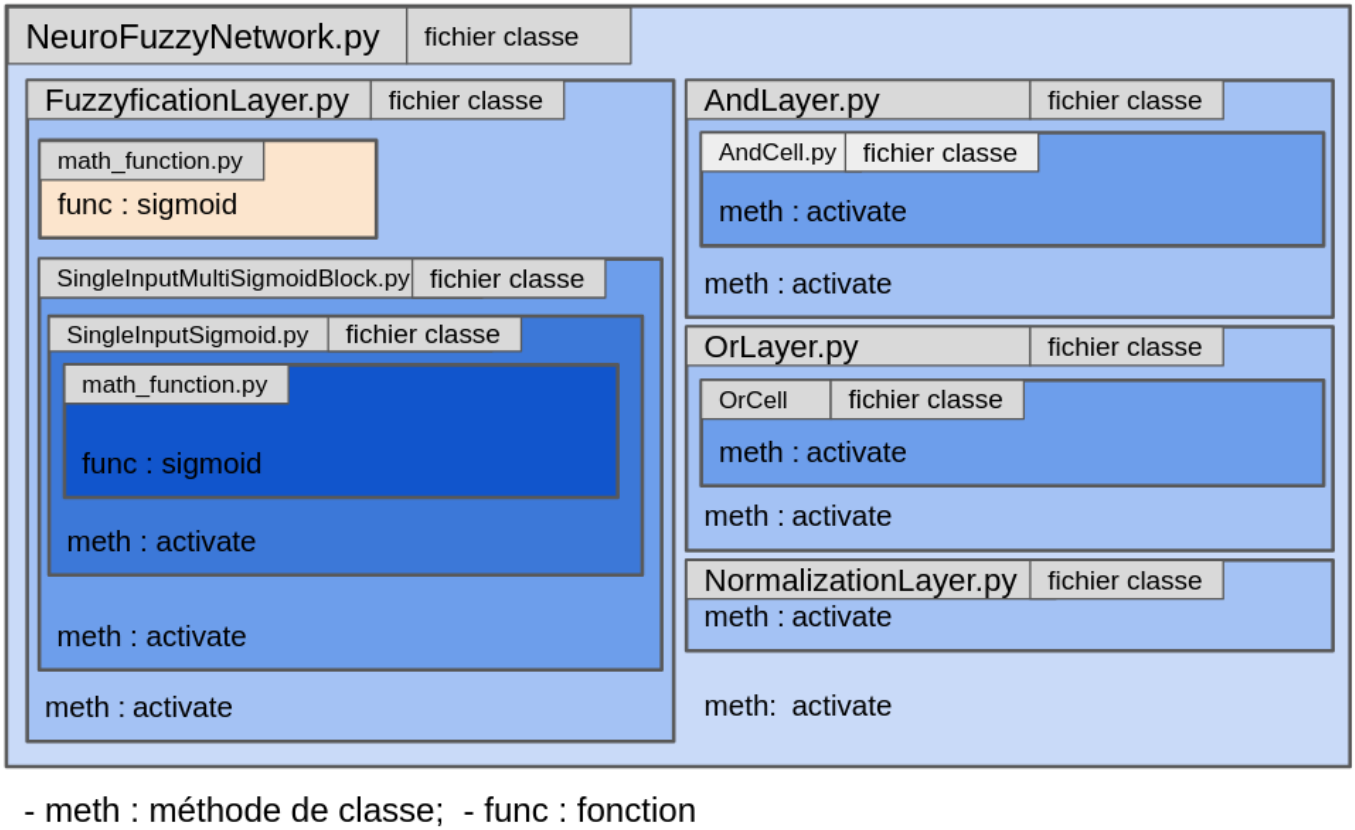


FIGURE 2.3 – Schéma des relations interclasses pour la partie réseau

La figure 2.3 résume l'ensemble des interactions entre les différentes classes avec les informations présentées jusqu'ici. Chaque classe a été implémentée dans un fichier .py éponyme. Les méthodes $\_\_init\_\_()$ intrinsèques à chaque classe ne sont pas représentées sur le schéma. Le fichier math_function.py possède simplement la fonction $sigmoid()$ qui permet de calculer la valeur d'une sigmoïde comme définie par (1.2). Des listes ont été utilisées tout au long de l'implémentation car il

s'agit d'une structure qui permet la construction dynamique du réseau. Dans la prévision des pistes d'amélioration du code, il pourrait être nécessaire de pouvoir ajouter ou supprimer des cellules en cours d'apprentissage.

Avec la répétition de fonctions sur plusieurs variables, une vectorisation aurait pu être intéressante (via numpy par exemple) mais manque de souplesse. Une implémentation en calcul parallèle plus tard peut être envisagée.

2.3.2 Calcul du gradient

L'implémentation des sensibilités effectuée lors du projet long de l'N7 n'a pas été reprise. Les différents calculs de sensibilités ont été implémentés dans chacune des couches associées. L'objectif étant d'obtenir un fonctionnement analogue à la passe *forward* pour la rétropropagation. Chaque couche du réseau fait appel à la couche suivante pour transmettre le calcul du gradient. Le calcul des sensibilités est implémenté dans la méthode *activate()* de chaque couche. Il peut être activé ou non à l'aide d'un argument booléen. Une nouvelle méthode *back_propagation()* a été implémentée dans chacune des couches pour le calcul du gradient.

Afin de s'assurer de la bonne implémentation de la méthode de calcul du gradient, un calcul numérique du gradient a également été implémenté en utilisant le taux d'accroissement. L'algorithme ne sera pas détaillé car accessoire dans le cadre de ce stage. Une différence relative inférieure à 10^{-4} a été observée par rapport au calcul analytique lors de la comparaison avec le calcul numérique. Au vu de la différence relative très faible obtenue, nous pouvons conclure que l'implémentation du calcul du gradient analytique a été effectuée correctement.

2.3.3 Présentation des algorithmes

Chaque composant du réseau a été réimplémenté dans des fichiers différents, afin de permettre notamment d'améliorer la clarté, et la réutilisabilité des fonctions et des classes lorsqu'elles peuvent être communes à plusieurs parties du code. La structure globale des couches et des cellules est implémentée sous forme de classes imbriquées. La classe cellule ne possède qu'une méthode *activate()* qui permet de calculer le résultat de cette cellule selon sa fonction (floutage, conjonction, ...). Chaque couche possède deux méthodes *activate()* et *back_propagation()* qui permettent respectivement de calculer de façon récursive la valeur d'une couche selon le score en entrée, et le gradient de cette couche. Une couche est composée d'une association de classes cellules. Chaque couche a été pensée afin d'avoir une taille adaptable aux données d'entrée et de sortie. De plus, nous retrouvons bien la structure en liste comme expliqué précédemment. Enfin, un booléen *train* permet de décider de calculer ou non la sensibilité nécessaire lors de la phase d'apprentissage.

2.4 Conclusion

Dans ce chapitre, nous avons pu présenter les révisions apportées au code issu du projet long de l'N7. Une révision très mineure de la partie théorique a été effectuée. La structure et l'organisation du code ont été revues afin de tirer parti au mieux de la Programmation Orientée Objet et permettre une réelle adaptabilité. Le format des données a également été revu et un nouveau jeu de données a été introduit, dû à la difficulté à séparer les différentes classes pour le jeu de données initial.

Chapitre 3

Algorithme d'apprentissage

L'apprentissage dans ce réseau s'articule selon deux grands axes : l'optimisation des poids et biais numériques, et l'optimisation des poids booléens. La couche de normalisation ne sera pas modifiée au cours de l'apprentissage car il s'agit uniquement d'une couche de formatage du résultat sous forme d'une distribution de probabilités.

Une description des relations entre fichiers est disponible figure 3.1, présentant la structure du code évoqué dans ce chapitre. Une nouvelle classe *TrainingTree* a été implémentée afin de regrouper les fichiers gérant les mécanismes de l'apprentissage hybride du réseau. Nous pouvons remarquer que la structure du réseau n'est construite qu'uniquement lors de l'optimisation numérique, *optim_num*. L'optimisation booléenne, *optim_bool*, ne fait qu'utiliser les informations transmises lors de l'optimisation numérique des différents réseaux. La façon dont nous gérons le stockage de ces informations sera évoquée à la fin du chapitre. Le fichier *converter.py* contient également deux fonctions servant lors du stockage des informations. Enfin, la méthode, *rzo_close()* permet d'explorer différentes configurations de réseau et sera expliqué dans ce chapitre.

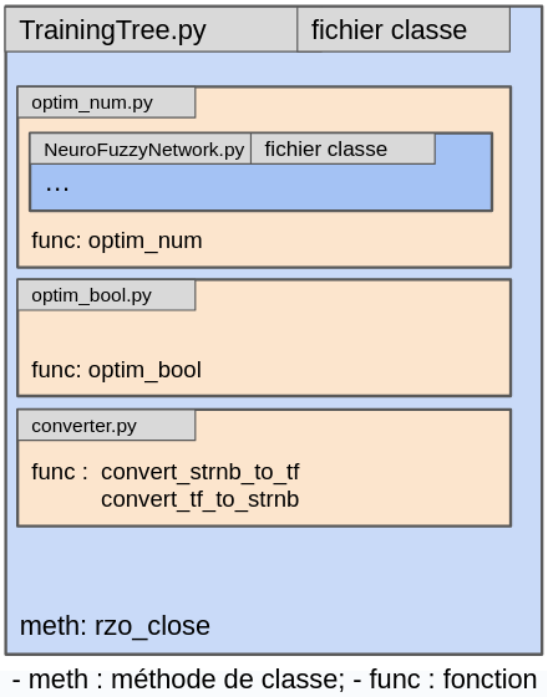


FIGURE 3.1 – Schéma des relations interclasses pour la partie réseau

Le réseau actuel n'étant encore qu'un prototype, nous utilisons une méthode de descente de

gradient classique à pas constant au cours l'optimisation numérique. L'objectif n'est pas pour l'instant de se concentrer sur l'efficacité de l'algorithme mais sur sa faisabilité. La fonction coût utilisée est la cross-entropie sans terme de pénalité donnée par l'équation 1.5.

3.1 Optimisation booléenne

L'optimisation des poids booléens consiste à pouvoir sélectionner, parmi une liste de réseaux potentiels, celui qui en prévision nous donnera le meilleur résultat. L'algorithme en pseudo-code est présenté sur l'algorithme 3. Pour chaque réseau, nous estimons la pente actuelle de la CE, et la fonction *PrevVal()* estime par une projection linéaire la valeur qu'il atteindrait une fois l'ensemble des itérations allouées à l'algorithme exploitées, dans l'hypothèse où l'intégralité de ces itérations lui étaient dédiées.

Algorithme 3 : Optimisation des poids booléens : recherche du réseau le plus prometteur (OptimBool)

```
Input : rzo_dict - Un dictionnaire contenant toutes les configurations de réseaux déjà
        considérées, ainsi que toutes les informations à leur sujet. Notamment : tendance
        actuelle de la CE (pente) du réseau et valeur actuelle de la CE (CE_actuelle).
Input : iter_rest - le nombre d'itérations restantes pour l'apprentissage
Output : optimal_id - le réseau le plus prometteur
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
```

L'estimation de la pente moyenne est effectuée de manière flottante : en tout temps, on estime la pente comme étant la moyenne sur un certain nombre des dernières itérations. Cette méthode a plusieurs avantages :

- Limitation et prévision de la taille de stockage nécessaire contrairement à une taille adaptative qui demanderait de stocker de plus en plus d'informations ;
- Limitation de l'influence à petite échelle : permet de lisser les irrégularités de la courbe pour représenter au mieux la tendance ;

— Limitation de l'influence à grande échelle : permet de se concentrer sur la performance courante d'un réseau à diminuer la fonction coût et ainsi de limiter l'impact sur le long terme d'une performance soudaine mais brève, généralement commune en début d'optimisation d'un réseau.

Comme expliqué précédemment, la performance d'un réseau est mesurée par projection de la CE sur le nombre restant d'itérations et non sur un temps infini, il est donc nécessaire d'estimer la valeur atteinte et non pas seulement de se contenter de la valeur de la pente.

On effectue une estimation linéaire de la valeur atteinte à l'aide de la fonction *PrevVal()*. La CE ne pouvant être négative, on considère deux cas lors de la sélection du meilleur réseau par l'algorithme 3 :

— Si le réseau estime une valeur négative, alors on prend en compte le nombre d'itérations qui lui est nécessaire afin d'annuler la cross-entropy, calculé par *PrevAbs()*

— Si le réseau estime une valeur positive, alors c'est cette valeur que l'on prend en compte pour la comparaison aux autres réseaux.

Un réseau qui estime une valeur négative sera toujours considéré plus performant qu'un réseau estimant une valeur positive. On s'assure de cette règle grâce au booléen *val_pos*, qui est mis à False dès qu'une valeur négative de CE est prévue, empêchant alors d'exécuter le code protégé par le deuxième test **if**.

Afin d'explorer de nouveaux réseaux, nous utilisons un algorithme, décrit en pseudo-code, cf. algorithme 4. Lorsqu'un réseau est optimisé numériquement, nous regardons également si ses voisins ont été considérés. Un réseau B est un voisin du réseau R si :

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

, où $\sim$ signifie que les poids sont à la même position dans le réseau et W_R^{bool} correspond à l'ensemble des poids du réseau neuroflou R.

Si les voisins du réseau n'ont pas été considérés, alors on ferme le réseau en parcourant tous les poids booléens et en générant une nouvelle configuration, via la fonction $Switch()$ qui change en son contraire un poids du réseau. Si le réseau n'a pas déjà été considéré, alors la fonction $OptimNum()$ estime sur un petit nombre d'itérations la valeur atteinte du nouveau réseau en l'optimisant dans le sens de la méthode de rétropropagation de gradient. Nous effectuons alors une optimisation booléenne comme décrite plus haut sur l'ensemble des voisins, contenu dans le *dictionnaire* (dans le sens d'un dictionnaire python) *voisin_dict*. Nous ajoutons uniquement le réseau le plus prometteur, *best_voisin*, à notre dictionnaire de réseaux.

Seul un voisin est ajouté car le nombre de réseaux à considérer peut facilement exploser (il y a $2^{nombre\_de\_poids\_booleens}$ réseaux potentiels). Afin de limiter également le nombre de voisin considérés, une méthode tabou est employée : une valeur d'ancienneté est attribuée à chaque poids du réseau, correspondant au nombre de réseaux ouverts depuis que ce poids a été modifié. Cette méthode favorise également l'exploration en contraignant l'algorithme à passer par des chemins qu'il n'aurait pas forcement pris car sous optimal au regard de l'ensemble des choix possibles à chaque étape.

Lors de la considération des voisins potentiels, un cas particulier doit également être appliqué à la dernière couche de poids booléens. En effet, afin de pouvoir prédire une classe, il est nécessaire qu'au moins un des poids soit à True. Ainsi, si un poids est à True sur la dernière couche, mais qu'il s'agit du seul, alors on ne considère pas le voisin du réseau qui mettrait à False ce poids, rendant de facto le réseau inutilisable.

Finalement, la condition $PositionCritique()$ correspond donc à :

— Le critère d'ancienneté n'est pas respecté

— Le poids se trouve sur une position critique sur la dernière couche

— La configuration est déjà présente dans l'ensemble des réseaux considérés

Algorithme 4 : Fermeture d'un réseau (FermetureRzo)

```
Input : rzo_parent - Un dictionnaire contenant toutes les informations du réseau à partir
        duquel nous envisageons les voisins.
Input : iter_rest - le nombre d'itérations restantes
Output : best_voisin - le réseau voisin le plus prometteur
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);
```

La fonction *Converter*() permet d'attribuer à chaque réseau un identifiant unique à partir de la configuration de ses poids booléens. La méthode sera détaillée dans une prochaine section de ce chapitre.

3.2 Processus d'optimisation générale du réseau

Afin d'optimiser ce réseau à poids mixtes, on alterne donc entre les phases d'optimisation des poids booléens et les phases d'optimisation de poids et biais numériques. Le schéma présenté figure 3.2 présente l'arbre de décisions à un instant t de l'apprentissage. Soit un réseau R considéré comme étant le plus prometteur sur le temps restant et dans la configuration actuelle, alors on commence par optimiser sur un nombre fixe *nbrep* d'itérations ses poids et biais numériques via un algorithme de rétropropagation. Une fois effectuée, si le réseau est ouvert, alors on le ferme à l'aide de la fonction $FermetureRzo()$, et sinon, on optimise alors les poids booléens en sélectionnant comme nouveau réseau, $meilleur\_rzo\_actuel$, celui qui nous donne le résultat le plus prometteur. La description complète en pseudo code est disponible sur l'algorithme 5.

Remarque : On commence d'abord par optimiser les poids et biais numériques avant les poids booléens dans un souci de cohérence entre le réseau initial et tous ceux considérés par évolution du voisinage.

3.3 Stockage

Afin de pouvoir passer efficacement d'un réseau potentiel à un autre, il est nécessaire de stocker différentes informations à leur sujet. Pour ce faire, nous établissons un dictionnaire imbriqué comme présenté figure 3.4. Chaque sous-dictionnaire est identifié par une clé unique. Cette clé unique correspond à la conversion décimale des poids booléens, concaténés et mis sous forme binaire. Une explication plus concrète du fonctionnement est disponible figure 3.3. Les fonctions $convert\_strnb\_to\_tf()$ et $convert\_tf\_to\_strnb()$ permettent respectivement de passer de la forme décimale à booléenne et inversement.

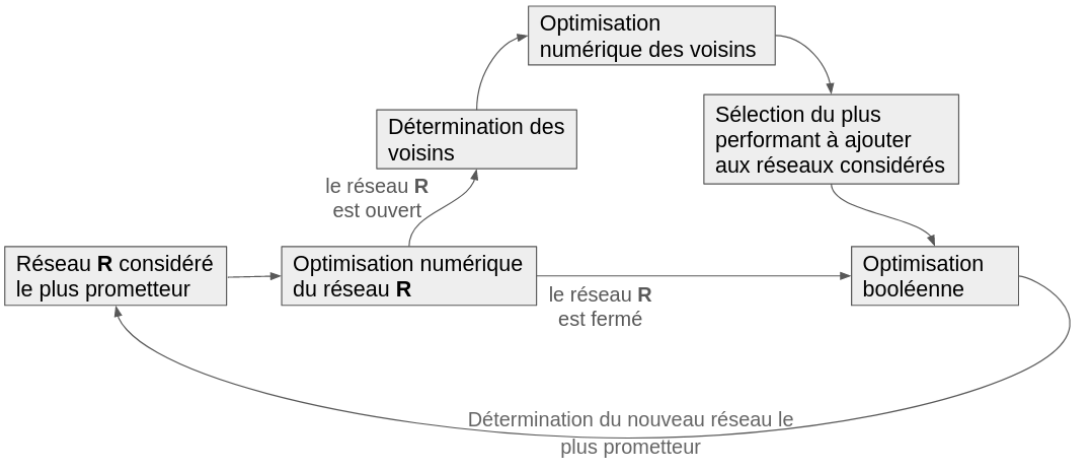


FIGURE 3.2 – Description de l'algorithme d'apprentissage des variables continues et booléennes

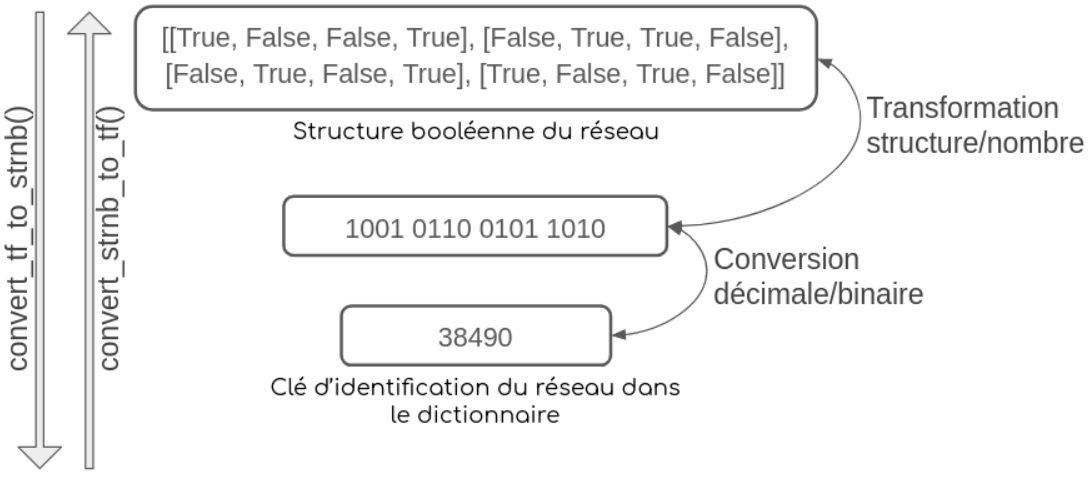


FIGURE 3.3 – Mécanisme d'attribution des clés lors du stockage

Algorithme 5 : Apprentissage avec des variables booléennes et continues

Input : rzo - Un réseau neuroflou à poids mixtes
Input : iter_dispo - le nombre d'étape de rétropropagation possible globalement
Input : nbrep - le nombre d'itérations de rétropropagation à effectuer chaque fois que les poids numériques sont optimisés.
Input : data - l'échantillon d'entraînement des données à modéliser.
Output : meilleur_rzo_actuel : le réseau neuroflou le plus prometteur actuellement avec des paramètres adaptés

```
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
```

Voici la description des différentes informations contenues dans un sous dictionnaire :

- poids_num : liste des poids numériques du réseau au dernier état considéré. Cette valeur est initialisée avec la valeur du réseau parent lors de l'ouverture
- biais_num : liste des biais numériques du réseau au dernier état considéré. L'initialisation est analogue à weights_num
- ce_evol : liste flottante des dernières valeurs d'entropie croisée obtenues lors de l'optimisation numérique du réseau. Elle permet d'estimer la pente de la CE du réseau
- pente : Estimation de la pente de la CE du réseau
- estOuvert : état du réseau, *False* si les voisins du réseau ont été considérés , *True* sinon.
- bool_seniority : liste de l'ensemble des valeurs d'ancienneté des poids booléens nécessaire lors de l'ouverture du réseau. Elle est définie via la valeur de son réseau parent, mettant à 0 l'ancienneté du poids qui a été modifiée par rapport au réseau parent, et en incrémentant de 1 tous les autres. Cette valeur n'est plus considérée une fois le réseau fermé.

Ce stockage permet de n'avoir qu'un seul réseau actif à la fois, et de pouvoir activer et désactiver un réseau de façon efficace sans perdre d'information.

3.4 Conclusion

Dans ce chapitre, nous avons pu présenter de façon détaillée le fonctionnement de la méthode d'optimisation mise au point dans le cadre d'un réseau de neurones comportant à la fois des variables booléennes et continues. La méthode se base sur une alternance de deux optimisations : - en optimisant numériquement de manière locale et monotone, c'est à dire uniquement le meilleur réseau ; - en optimisant les poids booléens de façon également locale mais non monotone (nous autorisons à

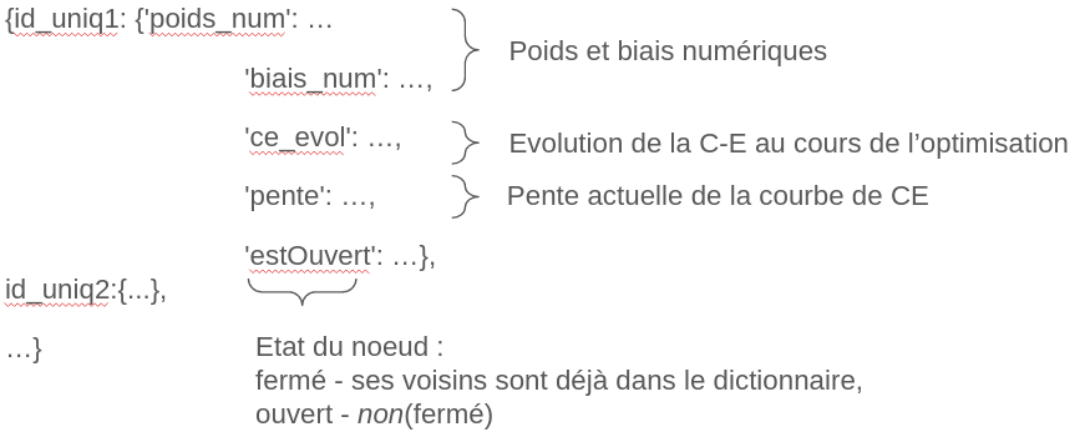


FIGURE 3.4 – Visualisation de la méthode de stockage des réseaux

considérer un voisin qui est moins bon que la solution courante) via la sélection du réseau optimal dans l'ensemble des réseaux considérés.

Nous avons aussi pu présenter la façon dont les différents réseaux sont stockés au cours de l'apprentissage.

Chapitre 4

Établissement de la règle de décision

Afin de rendre intelligible le réseau sélectionné après apprentissage, il est nécessaire de lui appliquer plusieurs transformations afin de traduire les différentes valeurs numériques et booléennes obtenues.

4.1 Transcription linguistique

Une première transformation consiste à traduire de manière naïve l'ensemble des poids et biais.

Poids et biais numérique : Notre fonction de floutage étant la fonction sigmoïde (comme définie dans 1.1), un rapide calcul nous montre que le quotient $-\frac{b}{w}$ donne la valeur en abscisse à laquelle la sigmoïde vaut 0.5, où w est le poids numérique associé à la sigmoïde et b le biais associé. Dans notre cas, nous ne considérons que les descripteurs *petit* et *grand* comme descripteur pour chacun des scores. Alors, dans le cas du descripteur *petit* , la valeur sera considérée petite, en dessous du quotient $-\frac{b}{w}$, et dans le cas *grand* , la valeur sera considérée grande au dessus de ce même quotient.

Poids booléen : Les poids booléens vont être traduits par la signification logique des différentes couches. Ainsi, la règle pour une classe donnée correspondra a une disjonction de conjonctions. Pour une classe donnée i, nous examinons les poids booléens $w_{i,j}^{disj}$, et pour chaque poids non nul, nous définissons un élément de la disjonction comme partie de la règle de cette classe. Cet élément est ensuite explicité en examinant les poids $w_{i,j}^{conj}$ pour le i correspondant à l'élément. La conjonction obtenue correspond alors à l'un des élements de la disjonction. Un fichier desc_data.py contient un dictionnaire avec les informations de chacun des scores et leur position dans la structure du réseau afin de pouvoir traduire par des mots les différents booléens non nuls. Le qualificatif *petit* ou *grand* est quant à lui déterminé en mesurant la direction de la pente.

4.2 Simplification de la règle

Afin d'avoir une règle réellement exploitable, il est nécessaire de simplifier la règle obtenue via l'opération précédente. En effet, la méthode d'exploration pour les poids booléens n'est pas intelligente et va seulement ajouter et supprimer des règles sans vérifier si des parties de la nouvelle configuration deviennent obsolètes. Ceci entraîne une agrégation importante de règles ne permettant par d'obtenir des règles pertinentes. Après l'utilisation de l'algorithme décrit ci-après, nous obtenons une utilisation moyenne de 2 scores pour définir une règle, contre 7 dans le cas avant simplification.

On définit la fonction *clean_ useless()*, qui pour un réseau considéré, va le simplifier. L'algorithme 6 présente son fonctionnement en pseudo-code.

Algorithme 6 : Simplification de la règle de décision

```
Input : rzo - Un réseau neuroflou à poids mixtes
Input : data - l'échantillon d'entraînement des données à modéliser.
Output : simplif_rzo : un réseau neuroflou équivalent en terme de performance mais dont
         les poids booléens sont maximisés à 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
```

Pour un réseau considéré, on va créer $rzo\_simplif$, une copie en valeur de ce réseau. Nous parcourons l'ensemble de ses poids booléens contenus dans $rzo\_simplif.poids\_bools$ et à chaque fois : si le poids est actif, le mettre à False. Alors, l'entropie croisée du réseau obtenu est calculée par la fonction $Cost()$ et nous la comparons à ce_ini l'entropie croisée du réseau initial. Si strictement aucune différence n'est observée, alors nous laissons le poids désactivé, sinon nous le réactivons en réutilisant $Switch()$.

Nous inversons l'ordre de parcours des couches grâce à la fonction $Reverse()$ afin de commencer par les couches les plus proches de la sortie. En effet, les couches sont inversement prioritaires dans la structure des règles par rapport à leur position dans le réseau. Ainsi, une disjonction peut être mise à False, rendant obsolète toute une conjonction. A l'opposé, une conjonction mise à False ne donne aucune information sur une disjonction.

Également, il n'y a pas unicité de la règle de simplification, car si deux paramètres apportent la même information, alors la simplification de l'un, ou de l'autre est possible et mènera donc à deux règles finales différentes. Cependant, la méthode de simplification employée, et l'ordre de parcours des booléens étant le même pour tous les réseaux, l'une des règles sera toujours favorisée et il n'est pas possible que deux réseaux qui pourraient posséder la même simplification se retrouvent en fait avec deux simplifications différentes.

4.3 Conclusion

Dans ce chapitre, nous avons présenté la stratégie utilisée afin de rendre un réseau après apprentissage intelligible et compréhensible par un être humain. Cette stratégie se base d'abord sur la traduction des différentes valeurs de poids et de biais, puis sur la simplification intelligente des règles obtenues par le réseau.

Chapitre 5

Résultats

Dans ce chapitre nous allons appliquer le réseau que nous venons de construire tout d'abord sur une simulation standard, avec des paramètres déterminés de façon empiriques au cours de développement afin d'observer le comportement de l'algorithme. Ensuite, nous allons faire varier chacun des paramètres afin de mesurer leurs influences sur le fonctionnement de l'algorithme et ses performances. Enfin, nous évoquerons les différentes pistes d'amélioration pouvant être explorées .

5.1 Archivage des sorties

Un travail important a été effectué lors de ce stage afin d'archiver automatiquement l'ensemble des informations lors de l'utilisation du programme afin de permettre de garder une trace de tout ce qui a été effectué. Chaque exécution du programme est donc sauvegardée et apporte ainsi plusieurs choses :

- La visualisation de l'évolution de la CE pour les réseaux considérés lors de chaque optimisation numérique, différenciés par couleur.
- La visualisation des fonctions d'appartenances du réseau après optimisation pour tous les scores d'un jeu de données considéré.
- La création d'un fichier texte contenant le temps d'exécution, le nombre de réseaux ouverts, l'ensemble de la configuration du programme lors de l'exécution, les valeurs des poids et biais en sortie, de l'entropie croisée, matrice de confusion sur l'échantillon d'apprentissage et de validation et enfin la valeur de poids booléens avant et après simplification.

Nous appelons ici *matrice de confusion*, la matrice correspondant aux moyenne des probabilités des différentes prévisions en sortie du réseaux par rapport à la sortie réelle.

5.2 Configuration de l'exécution standard

Lors de nos différents tests, nous utilisons le jeu de données "Breast Cancer Wisconsin (Diagnostic)". La configuration standard est présentée sur la table 5.1. Ces valeurs on été définies de façon empirique au cours de l'implémentation du programme.

Les paramètres d'entrée sont :

- Poids et biais initiaux : Il s'agit de l'ensemble des variables du réseau initial. Ils traduisent les informations de la règle experte, c'est à dire la règle que l'on considère a priori assez proche de la réalité.
- NBREP : Nombre d'itérations effectuées à la suite lors de l'optimisation numérique (rétropropagation), que ce soit pour le réseau le plus prometteur ou pour un de ses voisins.

NBREP	10
EPSILON	1
ITER_GLOB	20000
SENIORITY	46
CE_FOR_SLOPE	5

TABLE 5.1 – Configuration initiale des paramètres globaux

— EPSILON : Pas utilisé dans l'algorithme de descente de gradient (car utilisation d'un pas constant)
— ITER_GLOB : Nombre d'itérations du gradient maximal que va effectuer l'ensemble des réseaux. Il s'agit du critère d'arrêt. Les itérations effectuées lors de l'optimisation numérique des voisins afin de sélectionner le plus pertinent sont comptabilisées car non négligeables à l'échelle de l'optimisation.
— SENIORITY : Critère maximal d'ancienneté après lequel un poids booléen peut de nouveau être changé
— CE_FOR_SLOPE : Taille du tampon sur lequel on estime la pente de la CE. Le nombre d'itérations entre les deux valeurs extrêmes correspondra alors à $Ce\_for\_slope * Nbrep$

5.2.1 Règle initiale

Il est nécessaire de définir une règle experte initiale sur nos données. D'après la visualisation des différentes données comme rappelé figure 2.1, nous sommes capables de définir des droites approximatives de sorte que, de part et d'autre, chaque classe soit majoritaire. Nous pouvons également définir des lois via les poids booléens avec l'observation de cette même figure. Les valeurs sont présentées sur le Tableau 5.2. Les valeurs des poids booléens quant à eux sont représentés sur le Tableau 5.3

	Radius_m	Texture_m	Perimeter_m	Area_m	Concavity_m
w *grand*	1.0	1.0	1.0	1.0	1.0
b *grand*	-0.5	-0.5	-0.5	-0.3	-0.2
w *petit*	-1.0	-1.0	-1.0	-1.0	-1.0
b *petit*	0.5	0.5	0.5	0.3	0.2

TABLE 5.2 – Valeurs standard des poids et biais numériques du réseau initial

Cellule 1	True	False	True	False	True	False	True	False	True	False
Cellule 2	False	True	False	True	False	True	False	True	False	True
Cellule 3	False	False	False	False	False	False	False	False	False	False
Cellule 4	False	False	False	False	False	False	False	False	False	False

(a) Couche de conjonction

Cellule 1	False	True	Fase	False
Cellule 2	True	False	False	False

(b) Couche de disjonction

TABLE 5.3 – Valeurs standards des poids booléens du réseau initial

Ainsi les règles initiales pour les deux classes sont :

— B est prédite si Radius_m est petit (< 0.5), Texture_m est petit (< 0.5), Perimeter_m est petit (< 0.5), Area_m est petit (< 0.3) et Concavity_m est petit (< 0.2)
— M est prédite si Radius_m est grand (> 0.5), Texture_m est grand (> 0.5), Perimeter_m est grand (> 0.5), Area_m est grand (> 0.3) et Concavity_m est grand (> 0.2)

Nous pouvons remarquer que les cellules 3 et 4 de la couche de conjonction ne sont pas exploitées dans le réseau initial. En fait, elles vont permettent l'ajout de règles lors de l'apprentissage.

5.2.2 Premier résultat avec l'algorithme et la configuration standard

Avant d'essayer de comprendre l'influence de chacune des variables globales et de les affiner, nous avons exécuter le programme avec les paramètres standards afin d'avoir une base de comparaison.

Le tableau 5.4 présente les différents résultats numériques suite à cet apprentissage. Il est difficile d'avoir un avis sur le temps de calcul, car l'objectif du programme pour le moment n'est pas la rapidité de calcul. Cependant nous pouvons tout de même noter que le temps n'est pas absurdement long. Nous pouvons remarquer que, très peu de réseaux sont ouverts en comparaison au nombre de réseaux potentiels (nous rappelons qu'il y a $2^{nombre\_de\_variables\_booleennes}$ configuration possibles pour les poids booléens). Nous remarquons également qu'il y a bien une diminution de la CE lors de la phase d'apprentissage. De plus, le fait que la CE sur les données de validation soit du même ordre de grandeur que la CE sur les données d'apprentissage indique l'absence de sur-apprentissage.

Temps d'exécution	36'31"
Réseaux fermés	33
CE réseau initial	0.6527
CE données d'apprentissage	0.2184
CE données de validation	0.1882

TABLE 5.4 – Résultats numériques à l'exécution de l'algorithme dans la configuration standard

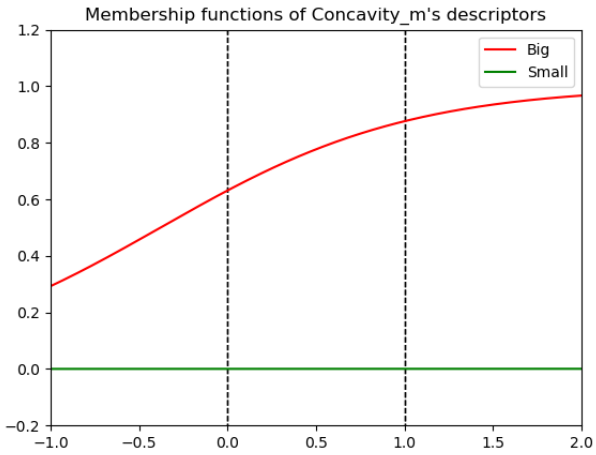
La règle décision déterminée après apprentissage est :
— B est prédite si Radius_m est petit (< -0.78)
— M est prédite si Concavity_m est petit (< -2.89)

Cette règle est loin d'être en accord avec l'intuition que nous avions initialement. D'autre part, nous pouvons directement conclure qu'elle n'est pas en accord avec ce que que nous avions prévu car la classe M est prédite à l'aide du qualificatif petit, là où, dans le cas du score Concavity_m, elle devrait être prédite à l'aide du qualificatif grand (cf 2.1).

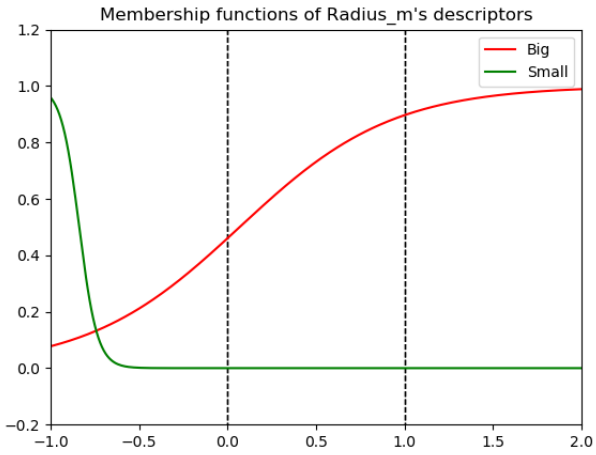
La figure 5.1, montre un comportement non souhaité par l'algorithme. En effet, nous aimerions avoir une couverture de l'espace importante entre $[0; 1]$ pour les critères qui sont utilisés dans les règles de décision. Cependant, on remarque que les deux critères utilisées dans les règles sont extrêmement faibles sur cet intervalle.

Ceci peut être dû à l'utilisation de la couche de normalisation en fin de réseau. En effet, cette dernière faisant un rapport de la couche précédente, il n'y a pas de contrainte à ce que les valeurs d'entrées soient hautes.

Nous pouvons tout de même constater sur la figure 5.2 que le comportement de l'algorithme n'est pas totalement en désaccord avec ce que nous souhaitons. Sur la première partie des itérations disponibles, l'exploration est favorisée et de nombreux réseaux différents sont considérés, et ceux qui ne sont pas prometteurs sont rapidement abandonnés. Plus le nombre d'itérations disponible diminue, plus l'algorithme se concentre sur de l'exploitation, en abandonnant au fur et à mesure les réseaux les moins bons. Sur les toutes dernières itérations, seul un réseau est exploité, il correspond bien à celui dont la est CE la plus basse.



(a) Score Concavity_m



(b) Score Radius_m

FIGURE 5.1 – Visualisation des fonctions d'appartenance des scores servant à établir les règles dans la configuration standard.

Comme nous allons le voir dans la suite, ces résultats sont relativement indépendants de la valeur des paramètres globaux. Ainsi, les problèmes pointés trahissent surtout le fait que cet algorithme est toujours en pleine conception et que ce rapport n'est qu'un rapport de mi-parcours.

5.3 Optimisation des paramètres d'entrée

Dans la suite, nous faisons varier les paramètres globaux afin de déterminer leurs influences et de vérifier des hypothèses que nous avons émises. L'utilisation de configurations limites nous permets également de s'assurer du bon fonctionnement de l'algorithme dans les cas "extrêmes" (valeurs très petites ou très grandes) et de s'assurer de son bon fonctionnement général.

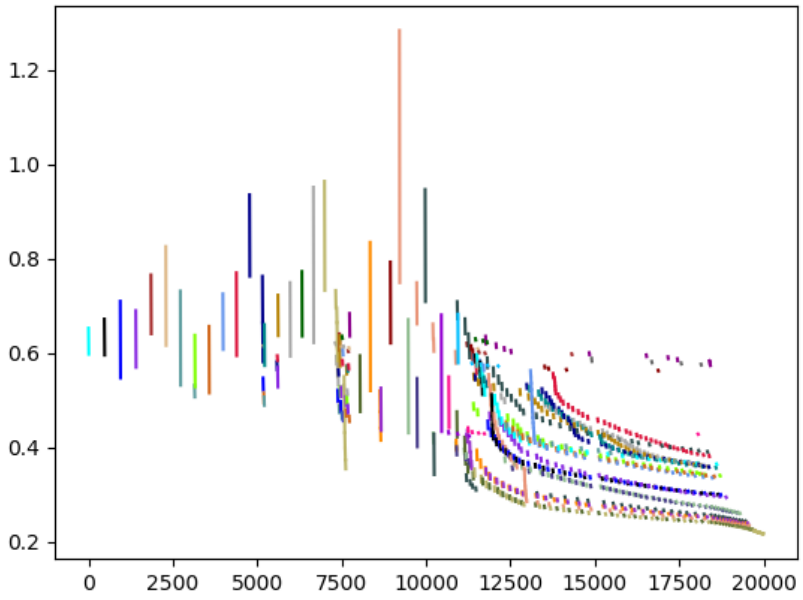


FIGURE 5.2 – Évolution de la CE sur le réseau considéré à chaque instant lors de l'apprentissage. Il s'agit de la superposition de la CE de tous les réseaux qui furent considérés comme "prometteurs" à un instant de l'apprentissage et que nous avons cherché à exploiter. En abscisse, le nombre d'itérations, en ordonné, la valeur de la CE

5.3.1 Poids et biais initiaux

Afin de déterminer l'influence de la règle experte, nous avons implémenté une fonction *rd_ weights()* qui permet de randomiser les poids et les biais numériques ou les poids booléens tout en conservant la structure souhaitée. Deux études ont ainsi été menées : premièrement nous avons fait varier les poids et biais numériques et deuxièmement nous avons fait varier uniquement les poids booléens.

Randomisation de l'ensemble des variables numériques

Lors de cette *randomisation* des poids et biais, nous prenons le parti de fixer leur signe. En effet, un changement de signe signifie un changement de la direction de descente pour la sigmoïde associée, soit un passage du descripteur *petit* à *grand* ou inversement. Or, nous souhaitons conserver la cohérence des descripteurs. De plus, le changement du type de descripteur sera traité dans l'expérience suivante. Les résultats ne sont pas représentés ici dans leur totalité, mais sont disponibles à nouveau au lien suivant. En effet, il n'y a pas de résultat pertinent qui puisse être tiré de cette expérience dans l'état du programme.

Nous pouvons quand même pointer que le nombre de réseaux considérés lors la phase d'apprentissage est très variable, entre 18 et 41. La CE de validation varie également autour de 0.22 avec un minimum à 0.16 et maximum à 0.35 sur l'ensemble des expériences. Enfin, les règles finales obtenues sont très différentes selon la configuration mais nous ne sommes pas parvenus à obtenir de modèle.

Randomisation de l'ensemble des poids booléens

Dans cette expérience, nous ne randomisons que les poids booléens, et nous considérons comme état initial pour les poids numérique un état moyen ($w = \pm 1, b = \mp 0.5$) qui correspond à une pente modérée et une fonction d'appartenance symétrique sur $[0; 1]$. Les poids numériques ne sont pas dans l'état standard comme présentés sur le tableau 5.2 car nous ne savons pas comment seront faites les connections avec la *randomisation* des poids booléens. De même que pour la première expérience, aucune tendance n'a pu être retirée de cette expérience. La CE de validation varie 0.20 à 0.31 et le nombre de réseaux ouverts de 22 à 61. De nombreuses règles différentes sont également observées.

Conclusion

Nous pouvons donc conclure partiellement que, la valeur initiale des poids et biais initiaux a bien une influence, à la fois sur la performance en terme de précision, mais également sur la règle finale établie après la phase d'apprentissage. De plus, la règle dite *experte* (cf tableaux 5.1), n'a pas été plus performante que l'ensemble des règles aléatoires. Nous pouvons tout de même noter que ses résultats sont plutôt bons sur l'échelle de l'ensemble des expériences avec randomisations en terme de performance. Il est cependant difficile de juger la pertinence des règles énoncées, ces dernières étant toutes fausses. C'est pourquoi nous n'avons pas été capables, en l'état, de faire ressortir des comportements pertinents du programme sur la base de ces expérimentations.

5.3.2 Ancienneté (*SENIORITY*)

Afin de déterminer l'influence du critère d'ancienneté de la méthode tabou, nous avons fait varier cette dernière de 0 jusqu'au maximum permis, c'est à dire le nombre de poids booléens du réseau. Nous pouvons d'ores et déjà constater qu'une valeur de zéro correspond à la non-utilisation de la méthode tabou. De même, aller à la valeur maximale (correspondant au nombre de poids booléens du réseau) signifie qu'en état stable, un seul poids ne pourra être modifié lors de l'ouverture d'un réseau. Au delà de cette valeur, il ne sera possible de faire varier qu'une unique fois chacun des poids du réseau.

Le critère d'ancienneté forçant le réseau à prendre des trajectoires qui ne sont, dans l'absolu, pas les plus optimales, nous nous attendons à voir le nombre de réseaux augmenter avec l'ancienneté. Ce critère permettant également de sortir des minimas locaux, nous nous attendons à obtenir de meilleurs résultats, à la fois en performance et en nature des règles apprises lorsqu'il augmente, sur un temps très long.

Expériences

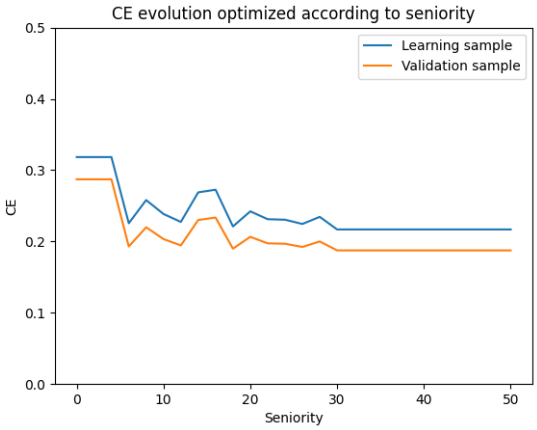
Les résultats sont présentés figure 5.3. On observe effectivement une amélioration non négligeable de la performance générale de l'algorithme avec l'utilisation de la méthode tabou. On peut remarquer également une stagnation de l'évolution à partir de Seniority = 30. Il est probable qu'à partir de cette valeur, les réseaux qui auraient altéré l'évolution du réseau se retrouvent inexploités car n'ayant pas dépassés le critère d'ancienneté. Ainsi, le même chemin est pris par le processus des poids booléens.

De manière générale, la règle de décision peut se résumer pour la plupart des cas à :

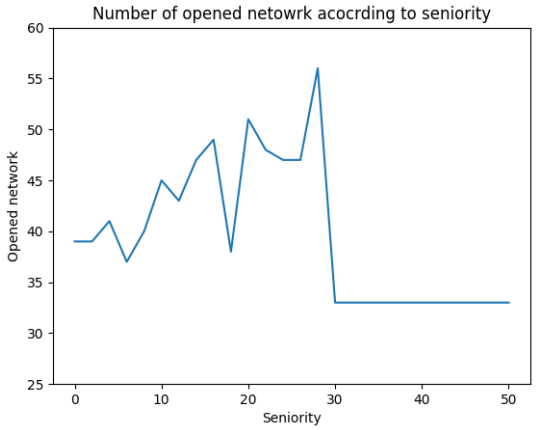
— B est prédite si Radius_m est petit ($<$ -0.78)

— M est prédite si Concavity_ est petit ($<$ -2.89)

Cette règle est difficile à interpréter, et est très proche de celle que nous avions pour le réseau standard. Les critères sont les mêmes, mais les valeurs de seuil sont légèrement différentes. Nous



(a) Evolution de la CE après optimisation



(b) Evolution du nombre de réseaux considérés

FIGURE 5.3 – Résultats après la variation du critère d'ancienneté

retrouvons également les problèmes que nous avions évoqués précédemment quelque soit la valeur du critère d'ancienneté.

On peut tout de même noter une même différence de règles pour le cas où l'ancienneté est inférieure à 4, sans pouvoir y apporter une justification, et sans qu'elle soit plus cohérente.

Conclusions

Ce que nous supposions avant ces expériences s'est avéré assez vrai. Il y a bien une amélioration de la CE et une augmentation du nombre de réseaux ouverts lorsque le critère d'ancienneté augmente. Cependant, un critère d'ancienneté trop grand est problématique pour l'exploration, qui dans notre cas a chuté drastiquement après un critère d'ancienneté supérieur à 30. Ce comportement problématique a pu être expliqué, et il est donc important de parvenir à trouver un équilibre lors du choix de la valeur du critère d'ancienneté afin de diminuer au maximum la CE après apprentissage, tout en s'assurant que l'exploration n'est pas impactée pour autant.

5.3.3 Nombre d'itérations global (*ITER_GLOB*)

Afin de déterminer l'influence de ce critère, nous avons conduit la même expérience avec quatre nombres d'itérations différents : 5000, 20000, 50000 et 200000, 500000. Cette expérience a pour objectifs de voir : le comportement de l'algorithme sur un temps très long, son comportement général avec l'augmentation du nombre d'itérations et sa sensibilité au sur-apprentissage.

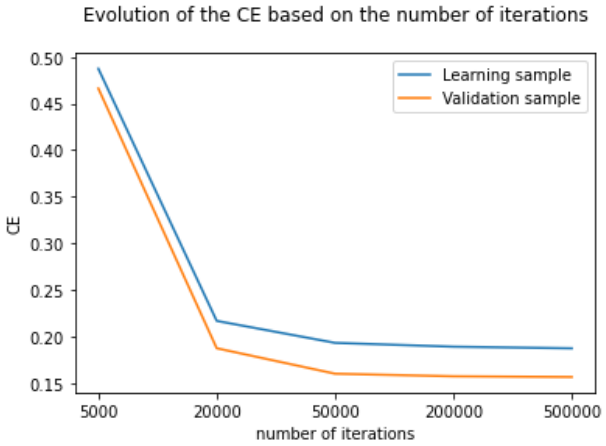
Expériences

Les résultats sont présentés figure 5.4. A la fois pour l'évolution de la CE et du nombre de réseaux considérés, on observe sur les premières expériences une nette amélioration. On observe cependant très peu de différence (voir aucune pour le nombre de réseau considérés) entre 200000 et 500000 itérations. Cependant, le temps de calcul (linéaire en nombre d'itérations), augmente énormément. Il n'est donc pas pertinent de faire l'apprentissage sur un nombre trop important d'itérations.

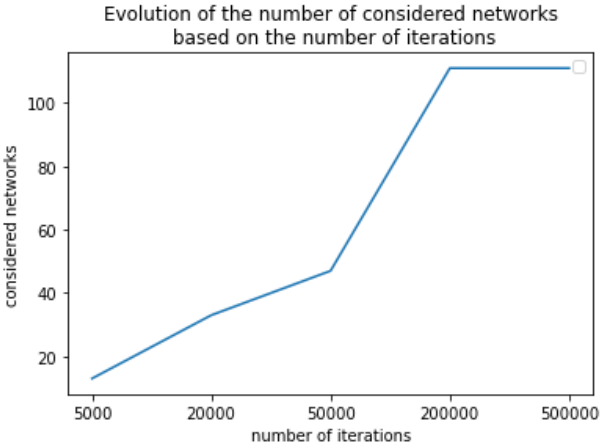
La règle établie pour seulement 5000 itérations n'est pas pertinente, ce nombre étant trop faible par rapport au problème et au jeu de données pour permettre une phase d'apprentissage. On retrouve le cas usuel pour 20000 itérations. A partir de 50000 itérations, la règle est :

— B est prédite si Area_m est petit (< -0.01)
— M est prédite si (Radius_ est petit (< -0.68)) ou (Concavity_m (> 0.56) est grand et Concavity_m est petit (< -0.32))

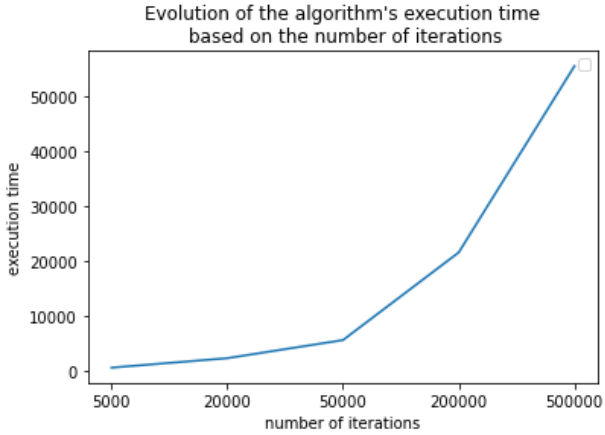
En plus d'avoir les mêmes problèmes que la configuration standard (les fonctions d'appartenance sont disponibles Annexe B.1), on observe également l'apparition de règles "contradictoires" que nous ne savons pas pour le moment gérer. Elles peuvent signifier plusieurs choses : -une mauvaise règle *de facto*, la nécessité de créer un descripteur *moyen* ou encore simplement une possibilité d'être à la fois grand et petit. Finalement, nous pouvons noter qu'aucun problème de sur-apprentissage n'est apparu pour des nombres d'itérations très grands



(a) Evolution de la CE après optimisation selon le nombre global d'itérations



(b) Evolution du nombre de réseaux considérés



(c) Evolution du temps d'exécution

FIGURE 5.4 – Résultats après la variation du nombre global d'itérations

REMARQUE : Sur la figure 5.4c, l'échelle en abscisse n'est pas linéaire.

Conclusion

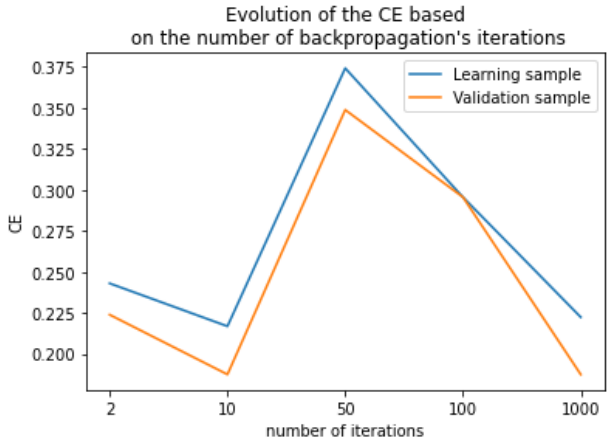
Cette expérience nous montre que dans l'état, l'utilisation d'un nombre d'itérations trop important n'est pas pertinent pour l'algorithme. A partir d'un certain rang, il n'y a aucune amélioration aussi bien sur l'exploitation que sur l'exploration. De plus, l'apparition des règles contradictoires, que l'algorithme ne sait pas gérer n'est pas souhaitable. Nous avons tout de même pu nous assurer de la robustesse de l'algorithme face au surapprentissage.

5.3.4 Nombre d'itérations lors de l'optimisation numérique (*NBREP*)

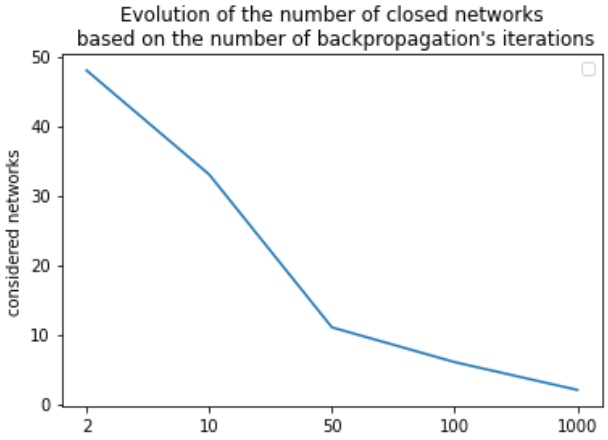
Afin de déterminer l'influence de ce critère, nous avons fait varier ce paramètre sur 5 valeurs : 2, 10, 50 et 100. Nous nous attendons à observer que, plus le nombre augmente, plus l'exploitation est favorisée au détriment de l'exploitation. A une valeur très faible (2) il est possible que l'algorithme soit inefficace et change trop régulièrement de réseau. En effet, le calcul de la pente de la CE étant lié à ce paramètre, plus ce dernier sera faible, plus l'algorithme sera sensible au très petites variations.

Expérience

Les résultats sont présentés figure 5.5. On observe une assez forte variation de la valeur de la CE sur la figure 5.5a. De façon étonnante à première vue, nous observons également que, un nombre d'itération extrêmement grand (1000), amène à une très bonne diminution de la CE. En observant la figure 5.5b, nous comprenons que, lors de cet apprentissage, seul deux réseaux ont été explorés, et qu'en fait, l'algorithme ne s'est concentré que sur l'exploitation au détriment de l'exploration. Une valeur trop grande n'est donc pas intéressante. Nous observons de manière générale sur la figure 5.5b que plus le nombre d'itérations de rétropropagation augmente, moins l'algorithme explore de nouveaux réseaux. Ceci est dû au fait que, lors de l'ouverture, un même nombre d'itérations est alloué à chaque voisin du réseau parent afin de l'exploiter, qu'il soit retenu, ou pas.



(a) Evolution de la CE après optimisation



(b) Evolution du nombre de réseaux considérés

FIGURE 5.5 – Résultats après la variation du nombre d'itérations lors de l'optimisation numérique

Conclusion

Il semble nécessaire d'avoir un certain équilibre dans le choix de cette valeur, afin de développer au mieux à la fois l'exploration et l'exploitation. Une valeur très grande peut sembler intéressante lorsque l'on regarde uniquement l'évolution de la CE. Cependant, la figure 5.5b nous montre que l'exploration n'est plus du tout prise en compte dans ce cas.

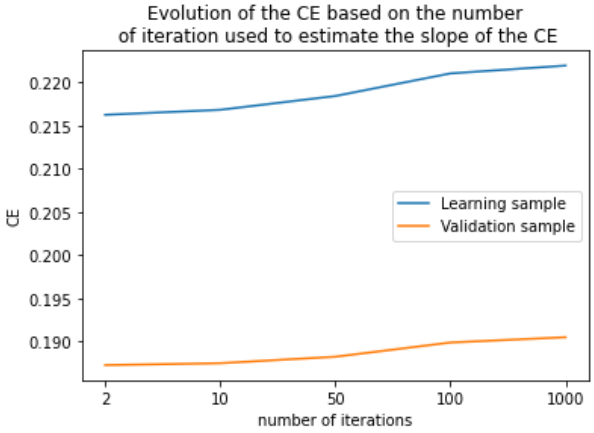
5.3.5 Taille du tampon (*CE_FOR_SLOPE*)

Afin de déterminer l'influence de ce critère nous l'avons fait varier sur 5 valeurs : 2, 5, 10 et 20 et 50. Une valeur de 2 correspond à ne prendre en compte que la dernière optimisation numérique pour le calcul de la pente (en plus de la valeur que nous venons d'obtenir lors de l'optimisation numérique)

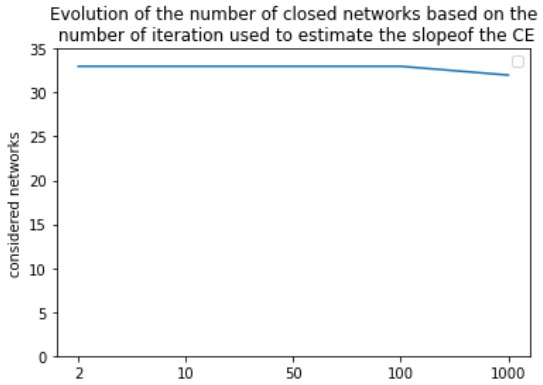
et de, finalement, ne pas considérer de tampon. Dans ce cas, on s'attend à avoir l'algorithme très sensible aux petites variations. Une valeur très grande (50), correspond à prendre, pour la plupart des réseaux, l'ensemble de leurs valeurs pour l'estimation de la pente. En effet, nous pouvons d'ores et déjà dire, grâce aux observations des expériences précédentes, que la plupart des réseaux ne sont pas exploités 50 fois par l'optimisation numérique. Nous nous attendons donc à avoir une favorisation des réseaux commençant à des valeurs très élevées, ces derniers ayant généralement une pente très abrupte au début.

Expériences

Les résultats sont présentés figure 5.6. On observe extrêmement peu de différences à la fois sur l'exploitation et sur l'exploration. La règle de décision après apprentissage est également la même ainsi que les seuils pour les descripteurs. On observe tout de même une différence sur l'exploitation des différents réseaux potentiels : il y a bien au cours de l'apprentissage, une favorisation de certains réseaux lorsque le nombre d'itérations augmente. Il s'agit de réseaux qui ont eu une forte décroissance au début. Les figures sont disponibles sur l'annexe B.3.



(a) Évolution de la CE



(b) Évolution du nombre de réseaux considérés

FIGURE 5.6 – Résultats après la variation du nombre d'itérations utilisées pour estimer la pente de la CE

Conclusion

Ce paramètre semble n'avoir que peu d'influence sur les résultats de l'algorithme. Une explication sur cette absence de sensibilité peut se trouver en partie dans l'allure de la CE. En effet, l'allure de cette dernière est assez régulière pour EPSILON = 1 (un exemple est disponible annexe B.2), ainsi aucun changement brusque ne peut modifier la pente estimé selon la taille du tampon. Nous avons tout de même remarqué un comportement qui favorise les réseaux ayant une grande progression sur leurs premières itérations lorsque la valeur augmente. Nous concluons donc qu'il faut prendre une valeur moyenne afin de ne pas être trop impacté par ce comportement, et également et ne pas être trop sensible aux petites variations éventuelles si la valeur était trop faible.

5.4 Voies d'améliorations

Comme nous l'avons expliqué, étant toujours à l'état de prototype, de nombreuses voies d'améliorations sont possibles pour ce réseau. Tout d'abord, plusieurs problèmes restent ouverts, le programme actuel ne permettant pas de répondre à nos exigences : les règles énoncés ne sont pas justifiables par l'homme. La mise en place d'une contrainte sur les descripteurs d'un même score afin de contrôler la couverture de l'espace de sorties par les fonctions d'appartenances est une idée que nous souhaitons mettre en oeuvre, dans l'espoir d'obtenir des règles et des seuils cohérents. Il est aussi nécessaire de statuer sur la gestion des règles contradictoires qui peuvent apparaître dans un processus de décision. Il pourrait aussi être intéressant de permettre d'allouer des nombres d'itérations différents lors de l'optimisation numérique, si celle-ci a lieu au cours de l'exploration de nouveau réseaux ou de l'exploitation d'un réseau déjà existant. Au niveau de l'efficacité de l'algorithme, l'amélioration de la descente de gradient ou l'introduction d'un pas adaptatif comme un pas de Wolfe [22], permettrait sans aucun doute d'obtenir une convergence plus rapide.

L'implémentation de méthodes de parallélisation permettrait également de pouvoir améliorer la vitesse, beaucoup de tâches étant répétitives : calculs matriciels, même algorithme appliqué à de nombreux scores différents, explorations des voisins, *etc.* Il pourrait également être intéressant de permettre au réseau, à partir d'une configuration initiale, d'enlever ou de rajouter des descripteurs linguistiques si ceux proposés ne sont pas suffisants. Enfin, la mise en place d'un standard complet et compréhensible par l'Humain afin de pouvoir suivre les processus de décisions en cours de d'apprentissage pourrait être intéressant.

5.5 Conclusion

Dans ce chapitre, nous avons pu observer le fonctionnement de notre algorithme. Ce dernier ne permet pas à ce stade de retourner des règles qui semblent cohérentes au vu des observations. Ce constat n'a rien de problématique : 'algorithme étant en développement, tous les problèmes identifiés sont en cours de résolution dans les versions suivantes de l'algorithme. Nous nous sommes également employés à faire des comparaisons monovariées des différents paramètres globaux du réseaux afin de déterminer leurs influences et d'étudier la réaction de l'algorithme à ces sollicitations. Suite à ça, nous avons pu proposer différentes piste d'améliorations.

Chapitre 6

Conclusions générales

6.1 Conclusion du sujet

Nous avons pu constater que l'explicabilité est une composante de plus en plus recherchée dans les algorithmes d'apprentissage, mais actuellement, aucun ne semble permettre de véritablement lever la problématique. Les réseaux de neurones de type neuro-flous semblent être une piste prometteuse et de nombreux chercheurs se penchent sur leur utilisation.

Le modèle de réseau neuro-flou développé est un modèle simple de perceptron à quatre couches, mais comportant à la fois des paramètres booléens et continus. Le modèle est volontairement simpliste car l'objectif est d'étudier la pertinence et la faisabilité de ce type de réseau.

Afin d'assurer l'apprentissage pour ce réseau hybride, une méthode spécifique a été mise au point et implémentée. Au cours de l'apprentissage, une alternance est effectuée entre une optimisation numérique via une rétropropagation et descente de gradient, et une optimisation booléenne par la création d'un arbre de configuration booléenne de réseaux.

Le projet est toujours en cours d'étude et les résultats intermédiaires, bien que ne permettant pas de retirer une règle intelligible, sont encourageant. La phase d'apprentissage et l'optimisation des poids booléens fonctionnent telle que nous l'avions souhaité. Les comportements problématiques pour le fonctionnement de l'algorithme ont pu être identifiés, comme la mauvaise couverture de l'espace de sortie par les fonctions d'appartenance. Des solutions ont pu être proposées et sont en cours d'étude sur la deuxième partie de mon stage.

6.2 Expérience personnelle

Au cours de ce stage, j'ai pu découvrir un peu plus le monde de la recherche. J'ai également pu étendre ma culture et mes connaissances mathématiques avec des domaines qui ne sont pas étudiés spécifiquement à l'INSA dans le parcours GMM : théorie des ensembles, logique floue, optimisation discrète, *etc.* J'ai également pu développer mes capacités de programmation dans l'objectif de rendre un produit utilisable et modulable.

Ce projet de recherche étant à ses débuts et théorique, de nombreuses pistes sont ouvertes à chaque identification de nouveau problèmes ou à l'émergence de nouvelles idées. J'ai donc été "contraint" d'adopter une forte organisation et une méthodologie efficace afin de ne pas me perdre parmi les différentes pistes et idées explorées en parallèle. Cette méthodologie passe par une organisation systématique du code, un archivage précis des différentes versions, et la prise en compte de l'archivage des résultats du réseau comme une composante importante du développement de celui-ci. J'ai également tenu un journal de stage, en décrivant systématiquement les expériences en cours, les modifications

apportées aux codes, les comptes rendus des réunions et les idées.

J'ai réussi à immédiatement m'investir et rentrer dans le vif du sujet rapidement. A contrario, la rédaction du rapport est de loin la partie qui m'a posé le plus de problèmes et un pan sur lequel il faut que je travaille si je souhaite effectuer une thèse à la suite de mon cursus d'ingénieur. Le problème vient essentiellement d'une difficulté à rester concentré sur une seule partie et ne pas m'éparpiller sur plusieurs parties en même temps.

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

Jeux de données

A.1 Moyennes et variances du jeu de données sur le crash d'avion sans échantillonnage stratifié

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 – Moyennes et variances du jeu de données sur le crash d'avion sans échantillonnage stratifié

A.2 Moyennes et variances du jeu de données sur le crash d'avion avec échantillonnage stratifié

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 – Moyennes et variances du jeu de données sur le crash d'avion avec échantillonnage stratifié

A.3 Moyennes et variances du jeu de données "Breast Cancer Wisconsin"

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

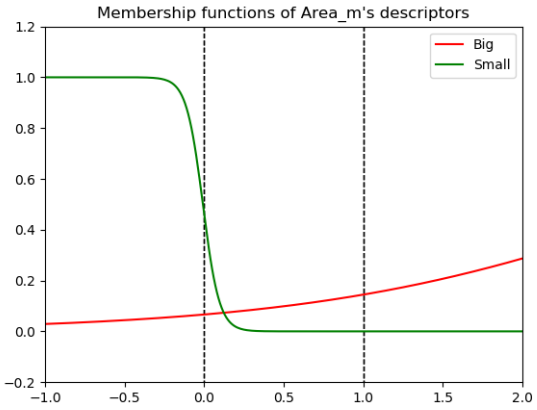
Classe M	moyenne	vairance	taille échantillon
Radius_m learning	0.616944	0.108601	148
Radius_m validation	0.631147	0.124161	64
Texture_m learning	0.546328	0.092707	148
Texture_m validation	0.558565	0.102679	64
Perimeter_m learning	0.607113	0.110494	148
Perimeter_m validation	0.623358	0.126092	64
Area_m 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 – Moyennes et variances du jeu de données "Breast Cancer Wisconsin"

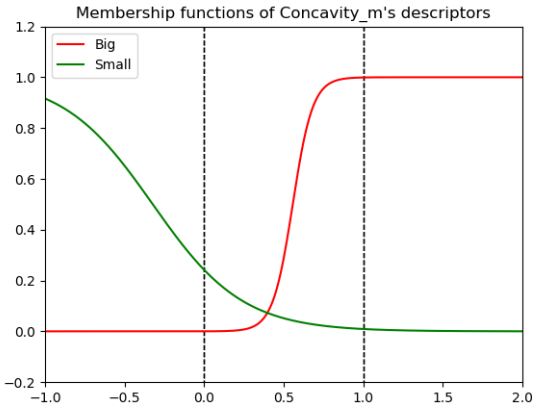
Annexe B

Résultats

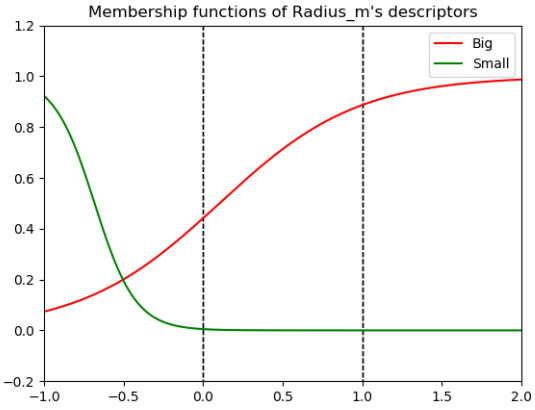
B.1 Fonctions d'appartenances



(a) Score Area_m



(b) Score Concavity_m



(c) Score Radius_m

FIGURE B.1 – Fonction d'appartenance pour les règles trouvées dans le à partir de 50000 itérations

B.2 Allure usuelle de la CE

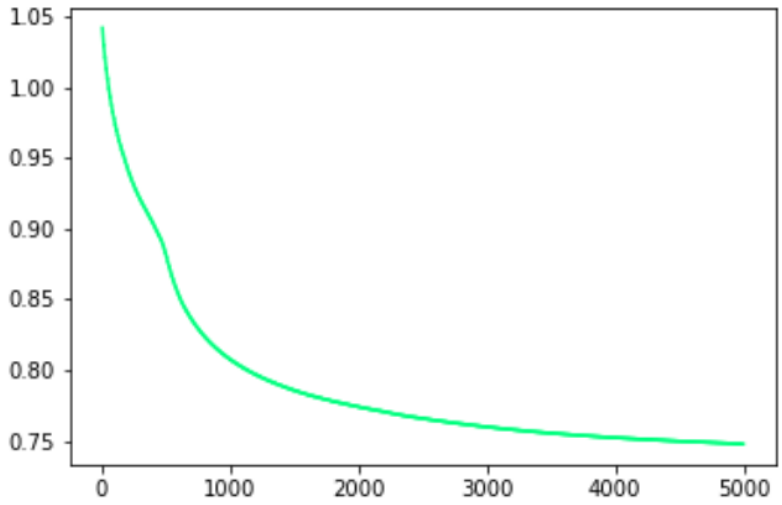


FIGURE B.2 – Allure usuelle de l'évolution de la CE pour un réseau quelconque dans le cadre de l'algorithme neuroflou à poids mixte

B.3 Graphiques d'affichage lors de la variation de la taille du tampon pour le calcul de la pente.

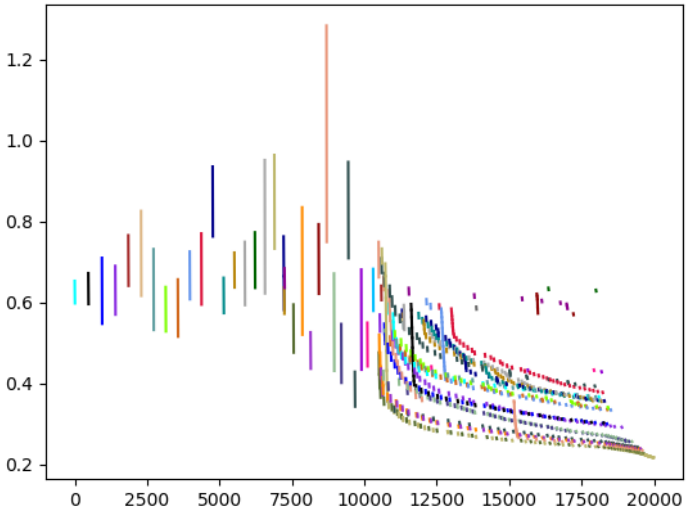


FIGURE B.3 – taille du tampon = 1*NBREP

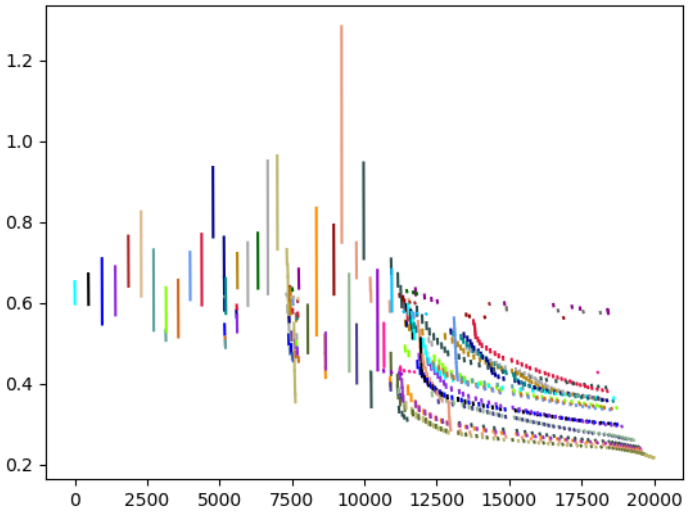


FIGURE B.4 – taille du tampon = 5*NBREP

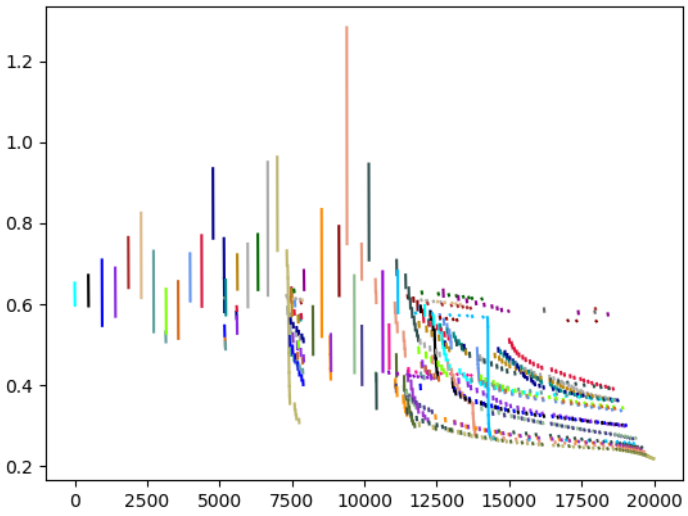


FIGURE B.5 – taille du tampon = 10*NBREP

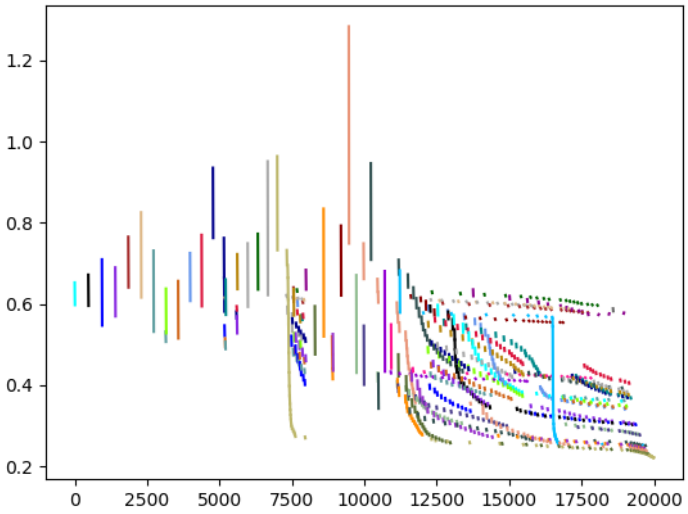


FIGURE B.6 – taille du tampon = 20*NBREP

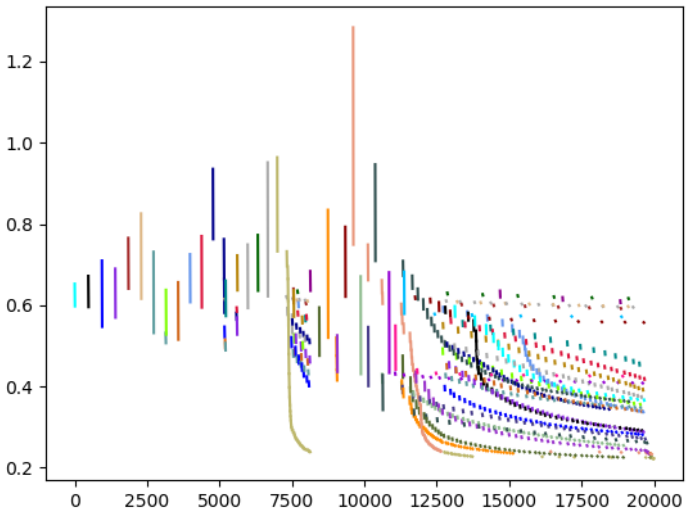


FIGURE B.7 – taille du tampon = 50*NBREP