
Étude expérimentale de l'algorithme Macsum

Ce document présente les résultats des expérimentations menées sur l'algorithme Macsum dans le cadre de l'apprentissage par agrégation, réalisées durant mon stage au LIP6 (CNRS, Sorbonne Université). Il contient des détails sur les expérimentations effectuées.

Auteur : Kinane Isaac

Encadrant : M. Marsala

Laboratoire : LIP6 (CNRS, Sorbonne Université)

Période : Juin 2025

Table des matières

1	Introduction et Cadre Expérimental	4
2	Formalisme de l’Agrégation Macsum	5
2.1	Définition des Notations	5
2.2	Équations des Bornes d’Intervalle	5
2.2.1	Borne Supérieure (\bar{y})	5
2.2.2	Borne Inférieure (\underline{y})	6
2.3	Fonctions de Perte Étudiées	6
2.3.1	Perte de base : Interval Mean Square Error (IMSE)	6
2.3.2	Perte Sigmoïde pour la Contenance et la Précision (SPIL)	6
2.4	Analyse de la Dynamique de la Fonction de Perte	7
3	Apprentissage et Optimisation	7
3.1	Algorithme d’Optimisation : Adam	7
3.2	Calcul du Gradient	8
3.3	Generation des données	8
4	Experimentation	9
4.1	Taille du Kernel	9
4.1.1	Le taux de bonne prediction	10
4.1.2	Taille de l’interval	11
4.2	Taille du jeu de données	12
5	Multi-layer Macsum	12
6	Hybride Multi-layer Macsum et Perceptron	17
6.1	Motivation et limites du modèle MacsumNet initial	17
6.2	Principe de l’arithmétique des intervalles	17
6.3	Propagation des intervalles dans les couches linéaires	17
6.4	Propagation à travers les fonctions d’activation	18
6.5	Architecture du modèle hybride final	18
7	Résultats Expérimentaux sur la Fonction de Friedman	19
7.0.1	Analyse des Architectures avec une Couche d’Entrée Macsum	19
7.0.2	Analyse des Architectures avec une Couche d’Entrée Perceptron	21
7.0.3	Synthèse des Résultats	25
8	Conclusion	26

9 Références bibliographiques **27**

10 Annexe **27**

1 Introduction et Cadre Expérimental

La quantification de l'incertitude est une problématique centrale en intelligence artificielle, particulièrement dans les tâches de régression où une prédiction ponctuelle peut masquer des risques importants. La régression par intervalles offre une solution élégante en prédisant une plage de valeurs plausibles. Ce rapport s'inscrit dans ce contexte et présente une étude expérimentale approfondie de l'algorithme **Macsum**, une méthode innovante de régression par intervalles proposée par Yasmine Hmidy, Agnès Rico et Olivier Strauss dans leur article «*Macsum Aggregation Learning*».

Fondé sur la théorie de l'intégration de Choquet, l'algorithme Macsum se distingue par son approche d'agrégation non linéaire qui dépend de l'ordre des caractéristiques d'entrée. L'objectif principal de ce stage a été de traduire cet algorithme théorique en un outil tangible et d'en explorer les dynamiques d'apprentissage à travers trois contributions :

1. **Implémentation et Validation de l'Opérateur Macsum** : La première étape a consisté à traduire l'algorithme Macsum en code fonctionnel. Une double implémentation a été réalisée : d'abord en **NumPy** pour une compréhension fine des mécanismes, puis en **PyTorch** pour l'intégrer à l'écosystème du deep learning.
2. **Conception d'une Architecture Hybride ('HybridNet')** : L'analyse des modèles 'Macsum' purs ayant révélé des défis de stabilité, une nouvelle architecture, '**HybridNet**', a été conçue. C'est la contribution centrale de ce travail. Elle fusionne :
 - Des **couches de perceptron** ('Linear') pour l'extraction de caractéristiques, rendues compatibles avec la tâche grâce à l'implémentation de l'arithmétique des intervalles.
 - Des **couches Macsum** agissant comme des agrégateurs spécialisés sur ces caractéristiques de haut niveau.
3. **Analyse Expérimentale des Dynamiques d'Apprentissage** : Une étude a été menée pour évaluer la performance de 'HybridNet'. Cette analyse a permis d'identifier et de résoudre des comportements d'optimisation complexes, comme l'effondrement de la borne inférieure, en ajustant la topologie du réseau et les hyperparamètres de la fonction de perte.

Ce document détaille la démarche suivie, du formalisme mathématique à l'analyse des résultats. L'ensemble du code source est accessible sur le dépôt GitHub : <https://github.com/Isaac-KD/Hybrid-Interval-Models>.

2 Formalisme de l'Aggrégation Macsum

L'algorithme Macsum est un modèle de régression par intervalles basé sur un vecteur de poids appris $\varphi \in \mathbb{R}^N$, que l'on appellera le *kernel*. Pour un vecteur d'entrée $x \in \mathbb{R}^N$, le modèle prédit un intervalle $[\underline{y}, \bar{y}]$ dont les bornes sont définies par les intégrales de Choquet discrètes, notées $\check{\mathbb{C}}_{\nu_\varphi}$ et $\check{\mathbb{C}}_{\nu_\varphi}^c$.

Intuitivement, l'algorithme fonctionne comme une somme pondérée, mais d'une manière non linéaire. Il trie d'abord les valeurs d'entrée de x par ordre croissant, puis il applique un poids à chaque "saut" de valeur. Ce poids n'est pas constant : il est calculé dynamiquement en fonction de l'importance des caractéristiques restantes, telle que définie par le kernel φ .

2.1 Définition des Notations

Pour comprendre les équations, définissons d'abord chaque terme :

- N : Le nombre de caractéristiques (la dimension du vecteur d'entrée x).
- $x = (x_1, \dots, x_N)$: Le vecteur des caractéristiques d'entrée.
- $\varphi = (\varphi_1, \dots, \varphi_N)$: Le kernel, un vecteur de poids appris par le modèle.
- $x_{(k)}$: La k -ième plus petite valeur du vecteur x . Par exemple, si $x = (8, 2, 5)$, alors $x_{(1)} = 2$, $x_{(2)} = 5$, et $x_{(3)} = 8$. On définit $x_{(0)} = 0$ par convention.
- φ^+ et φ^- : La décomposition du kernel en sa partie positive et négative.
 - $\varphi_i^+ = \max(0, \varphi_i)$
 - $\varphi_i^- = \min(0, \varphi_i)$
- $\varphi_{(i)}$: C'est le terme le plus important. Il ne s'agit pas du i -ème élément de φ , mais de l'élément de φ qui correspond à la caractéristique $x_{(i)}$. Autrement dit, si x_j est la i -ème plus petite valeur, alors $\varphi_{(i)} = \varphi_j$. Le kernel φ est donc **permuté selon le même ordre que le vecteur d'entrée x** .

2.2 Équations des Bornes d'Intervalle

Avec ces notations, les bornes de l'intervalle sont calculées comme suit :

2.2.1 Borne Supérieure (\bar{y})

La borne supérieure est calculée par la somme des contributions positives maximales et négatives minimales.

$$\bar{y} = \check{\mathbb{C}}_{\nu_\varphi}(x) = \sum_{k=1}^N (x_{(k)} - x_{(k-1)}) \cdot \left(\max_{i=k}^N \varphi_{(i)}^+ + \min_{i=k}^N \varphi_i^- + \min_{i=k}^N \varphi_{(i)}^+ \right) \quad (1)$$

À chaque étape k de la sommation, le poids appliqué à l'incrément de valeur $(x_{(k)} - x_{(k-1)})$ est la somme du plus grand poids positif et du plus petit poids négatif (le plus négatif) parmi tous les éléments restants (de l'indice k à N) dans le kernel permuté.

2.2.2 Borne Inférieure (\underline{y})

La borne inférieure est calculée de manière duale, en combinant les contributions positives minimales et négatives maximales.

$$\underline{y} = \check{\mathbb{C}}_{\nu_\varphi}^c(x) = \sum_{k=1}^N (x_{(k)} - x_{(k-1)}) \cdot \left(\max_{i=k}^N \varphi_{(i)}^+ + \min_{i=k}^N \varphi_i^- + \min_{i=k}^N \varphi_{(i)}^+ \right) \quad (2)$$

Ici, le poids est la somme du plus petit poids positif (potentiellement zéro) et du plus grand poids négatif (le moins négatif, potentiellement zéro) parmi les éléments restants du kernel permuté.

Cette structure duale garantit, par construction, que $\underline{y} \leq \bar{y}$.

2.3 Fonctions de Perte Étudiées

2.3.1 Perte de base : Interval Mean Square Error (IMSE)

La fonction de perte la plus simple est une extension de l'erreur quadratique moyenne, pénalisant la distance entre la cible y_j et les deux bornes de l'intervalle :

$$\mathcal{L}_{IMSE} = \sum_{j=1}^M \left((y_j - \underline{y}_j)^2 + (y_j - \bar{y}_j)^2 \right) \quad (3)$$

2.3.2 Perte Sigmoïde pour la Contenance et la Précision (SPIL)

Pour mieux contrôler l'équilibre entre la **contenance** (y_j doit être dans l'intervalle) et la **précision** (la largeur de l'intervalle), une fonction de perte alternative, nommée *Sigmoid Penalized Interval Loss (SPIL)*, a été testée. Soit les écarts :

$$\underline{w}_j = \underline{y}_j - y_j, \quad \bar{w}_j = y_j - \bar{y}_j \quad (4)$$

La perte SPIL est définie par :

$$\mathcal{L}_\sigma = \sum_{j=1}^M \left[\alpha \cdot (\bar{y}_j - \underline{y}_j) + \gamma \cdot (\sigma_k(\bar{w}_j) \cdot \bar{w}_j^2 + \sigma_k(\underline{w}_j) \cdot \underline{w}_j^2) \right] \quad (5)$$

où $\sigma_k(x) = (1 + e^{-k \cdot x})^{-1}$ est la fonction sigmoïde. Ici, α pondère l'importance de la largeur de l'intervalle, tandis que γ pénalise les erreurs de contenance. Une forme asymétrique avec $\bar{\gamma}$ et $\underline{\gamma}$ a également été considérée.

2.4 Analyse de la Dynamique de la Fonction de Perte

Problématique rencontrée : Lors des expérimentations, il a été observé que, sans normalisation des données, le terme d'erreur quadratique dans la perte SPIL domine largement le terme de largeur de l'intervalle (pondéré par α). Par conséquent, l'optimiseur par descente de gradient se concentre d'abord sur la minimisation des erreurs (assurer que $y_j \in [\underline{y}_j, \bar{y}_j]$), avant de commencer, dans un second temps, à réduire la largeur des intervalles.

Variante testée : Pour adresser ce problème d'échelle, une variante pénalisant le carré de la largeur de l'intervalle a été testée :

$$\mathcal{L}_{\sigma, \text{quad}} = \sum_{j=1}^M \left[\alpha \cdot (\bar{y}_j - \underline{y}_j)^2 + \gamma \cdot (\sigma_k(\bar{w}_j) \cdot \bar{w}_j^2 + \sigma_k(\underline{w}_j) \cdot \underline{w}_j^2) \right] \quad (6)$$

Bien que cette approche réduise la sensibilité à l'échelle des sorties, elle a complexifié le réglage des hyperparamètres, rendant l'optimisation globalement moins stable.

Choix final : En conséquence, la formulation \mathcal{L}_σ a été retenue, offrant le meilleur compromis entre contrôlabilité et performance.

3 Apprentissage et Optimisation

3.1 Algorithme d'Optimisation : Adam

Pour résoudre le problème d'optimisation, nous avons utilisé l'algorithme **Adam** (Adaptive Moment Estimation), une méthode d'optimisation stochastique efficace basée sur le calcul des moments du gradient. La mise à jour des paramètres θ (ici, le kernel φ) suit la règle :

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \quad (7)$$

où :

- $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla_{\theta_t} \mathcal{L}_t$: estimation du premier moment (moyenne mobile des gradients).
- $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla_{\theta_t} \mathcal{L}_t)^2$: estimation du second moment (moyenne mobile des gradients au carré).
- \hat{m}_t et \hat{v}_t sont les versions corrigées pour le biais initial.

3.2 Calcul du Gradient

La descente de gradient nécessite le calcul de $\nabla_\varphi \mathcal{L}$. Pour la perte **IMSE**, le gradient est direct :

$$\nabla_\varphi \mathcal{L}_{IMSE} = 2 \cdot \sum_{j=1}^M \left((\underline{y}_j - y_j) \cdot \nabla_\varphi \underline{y}_j + (\bar{y}_j - y_j) \cdot \nabla_\varphi \bar{y}_j \right) \quad (8)$$

Pour la perte **SPIL**, la règle de la chaîne donne un gradient plus complexe :

$$\nabla_\varphi \mathcal{L}_\sigma = \sum_{j=1}^M \left(\frac{\partial \mathcal{L}_{\sigma,j}}{\partial \bar{y}_j} \nabla_\varphi \bar{y}_j + \frac{\partial \mathcal{L}_{\sigma,j}}{\partial \underline{y}_j} \nabla_\varphi \underline{y}_j \right) \quad (9)$$

avec les dérivées partielles par rapport aux bornes données par :

$$\begin{aligned} \frac{\partial \mathcal{L}_{\sigma,j}}{\partial \bar{y}_j} &= \alpha - \gamma (\bar{w}_j^2 \cdot \sigma'_u + 2 \cdot \bar{w}_j \cdot \sigma_u) \\ \frac{\partial \mathcal{L}_{\sigma,j}}{\partial \underline{y}_j} &= -\alpha + \gamma (\underline{w}_j^2 \cdot \sigma'_l + 2 \cdot \underline{w}_j \cdot \sigma_l) \end{aligned}$$

et où les termes $\sigma_l, \sigma_u, \sigma'_l, \sigma'_u$ sont définis comme précédemment. Les gradients des bornes par rapport au kernel, $\nabla_\varphi \underline{y}$ et $\nabla_\varphi \bar{y}$, sont non triviaux et calculés conformément à l'approche de l'article de référence.

3.3 Génération des données

Pour la génération des données, nous avons généré un noyau (*kernel*) aléatoirement selon une loi uniforme dans l'intervalle $[-1, 1]^N$ et $[-10, 10]^N$, avec $N \in \{2, 20, 100\}$.

Les données ont été générées selon une distribution normale multivariée, indépendante dans chaque dimension :

$$X_i \sim \mathcal{N}(\mu, \sigma^2 I_N), \quad \text{où } \mu = 50 \cdot \sqrt{\pi} \cdot \mathbf{1}_N, \quad \sigma = \frac{50}{3}.$$

Pour chaque $x \in X$ on lui associe une cible y , $\forall i, y_i = \frac{\tilde{C}_{\nu\varphi}(x_i) - \tilde{C}_{\nu\varphi}^c(x_i)}{2} + z_i$ où z_i est un bruit pour que les données ne soient pas toutes centrées mais toujours dans l'intervalle généré par le kernel initial que l'on appellera le vrai kernel.

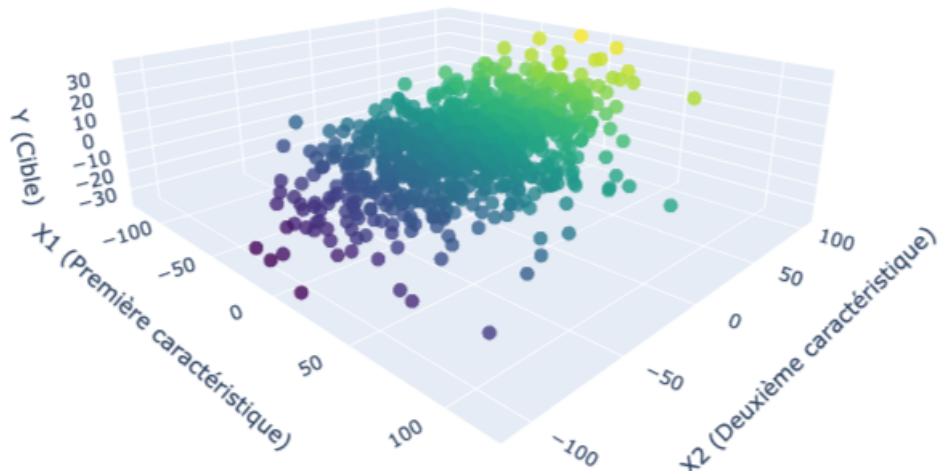


FIGURE 1 – Visualisation des données, N=2

4 Experimentation

4.1 Taille du Kernel

D'après les résultats obtenus au cours des différentes expérimentations, on observe deux phénomènes :

1. La loss IMSE est plus précise (meilleur taux de bonne prédiction) que la loss SPIL, mais cela s'inverse lorsque l'on augmente la taille de N (dimension du kernel) sans augmenter la taille du jeu de données.
2. L'intervalle de prédiction est plus petit pour la loss IMSE que pour SPIL.
Pour plus de détails, une annexe est disponible.

4.1.1 Le taux de bonne prediction

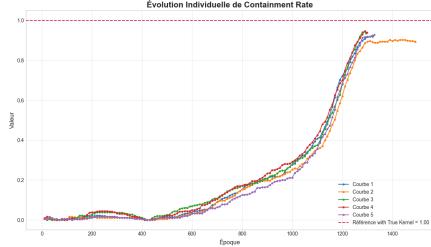


FIGURE 2 – Loss : SPIL, jeu de données de 2000 (taux de bonne prédiction)

Kernel de taille (N) : 2

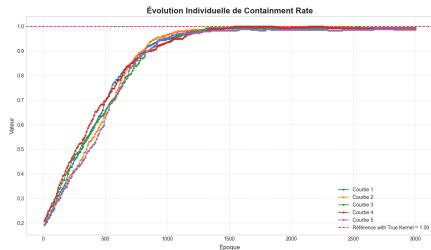


FIGURE 4 – Loss : SPIL, jeu de données de 2000 (taux de bonne prédiction)

Kernel de taille (N) : 20

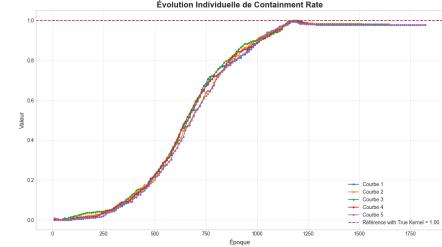


FIGURE 3 – Loss : IMSE, jeu de données de 2000 (taux de bonne prédiction)

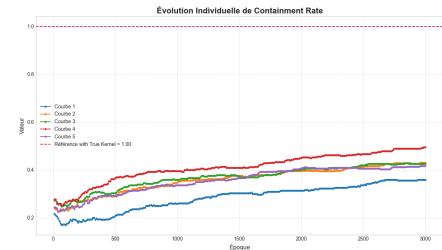


FIGURE 5 – Loss : IMSE, jeu de données de 2000 (taux de bonne prédiction)

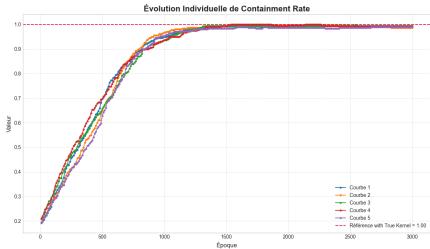


FIGURE 6 – Loss : SPIL, jeu de données de 2000 (taux de bonne prédiction)

Kernel de taille (N) : 100

4.1.2 Taille de l'intervalle

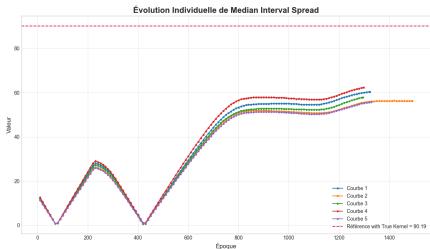


FIGURE 8 – Loss : SPIL, jeu de données de 2000 (taux de bonne prédiction)

Kernel de taille (N) : 2

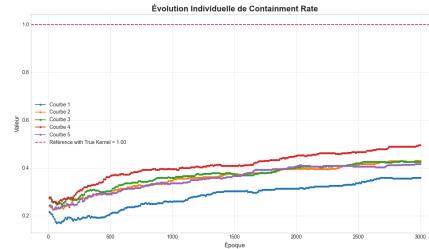


FIGURE 7 – Loss : IMSE, jeu de données de 2000 (taux de bonne prédiction)

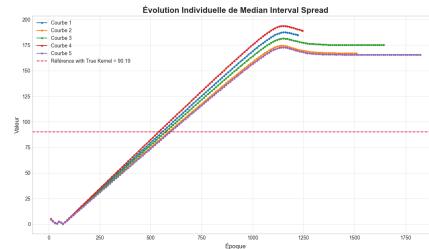


FIGURE 9 – Loss : IMSE, jeu de données de 2000 (taux de bonne prédiction)

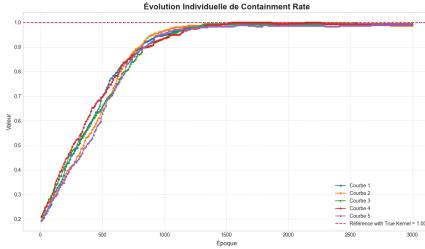


FIGURE 10 – Loss : SPIL, jeu de données de 2000 (taux de bonne pré-diction)

Kernel de taille (N) : 20

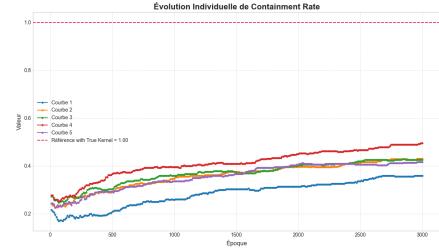


FIGURE 11 – Loss : IMSE, jeu de données de 2000 (taux de bonne pré-diction)

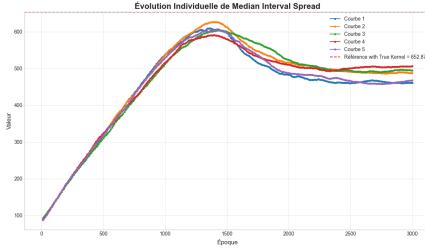


FIGURE 12 – Loss : SPIL, jeu de données de 2000 (taux de bonne pré-diction)

Kernel de taille (N) : 100

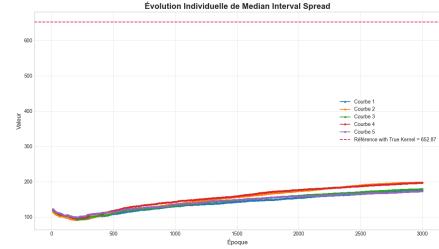


FIGURE 13 – Loss : IMSE, jeu de données de 2000 (taux de bonne pré-diction)

4.2 Taille du jeu de données

On observe sur les différentes expérimentations que plus les données augmentent, plus l'IMSE permet d'être précis.

5 Multi-layer Macsum

Nous avons également expérimenté une manière d'empiler plusieurs modèles Macsum, à la manière d'un réseau *Multi-layer Macsum*. La procédure adoptée est

la suivante :

Nous partons d'un vecteur d'entrée $x \in \mathbb{R}^N$, que nous dupliquons pour former un tuple $[x, x]$. Chaque copie du vecteur est ensuite associée à une borne (inférieure ou supérieure).

Ce tuple $[x, x]$ est transmis à chaque neurone de la première couche. Chaque neurone renvoie un intervalle, c'est-à-dire une borne inférieure et une borne supérieure. Si la couche contient n neurones, nous obtenons n intervalles : soit n bornes inférieures et n bornes supérieures.

On forme alors deux vecteurs de taille n :

$$\underline{y} = [\underline{y}_1, \underline{y}_2, \dots, \underline{y}_n], \quad \bar{y} = [\bar{y}_1, \bar{y}_2, \dots, \bar{y}_n]$$

On répète la même opération avec la seconde copie du vecteur d'entrée. On obtient ainsi deux tuples :

$$(\underline{y}^{(1)}, \bar{y}^{(1)}) \quad \text{et} \quad (\underline{y}^{(2)}, \bar{y}^{(2)})$$

Nous utilisons ensuite la borne supérieure calculée à partir du vecteur d'entrée associé à la borne supérieure (c'est-à-dire $\bar{y}^{(2)}$), et la borne inférieure calculée à partir du vecteur d'entrée associé à la borne inférieure (c'est-à-dire $\underline{y}^{(1)}$). Ces deux vecteurs forment un nouveau tuple $(\underline{y}^{(1)}, \bar{y}^{(2)})$. Nous appliquons alors une fonction d'activation à ce tuple, et le résultat est utilisé comme entrée pour la couche suivante.

Le schéma ci-dessous illustre ce fonctionnement :

- Les flèches **rouges** représentent les vecteurs associés aux bornes supérieures,
- Les flèches **bleues** correspondent aux vecteurs associés aux bornes inférieures,
- Les **cercles noirs** symbolisent les modules Macsum appliqués à chaque couche.

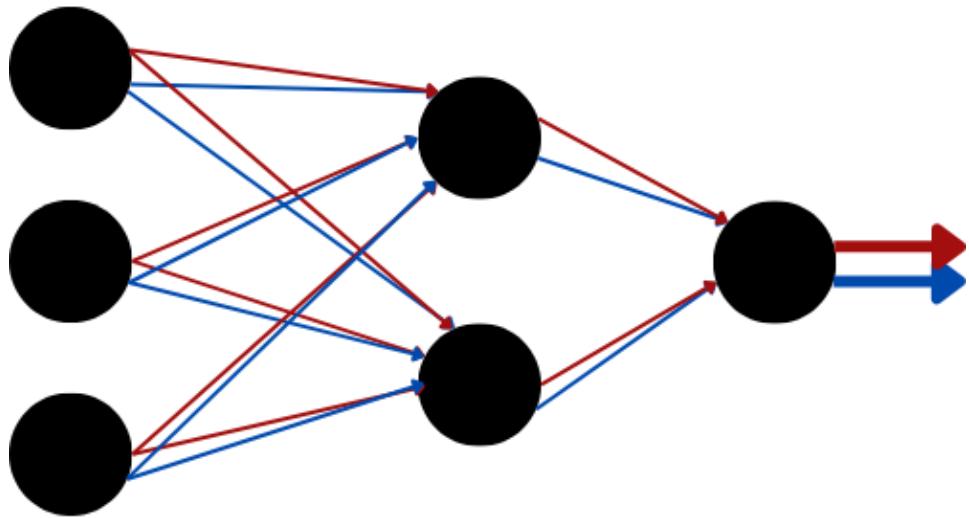


FIGURE 14 – Shema du Multi- layer - Macsum

Pour expérimenter les performances du modèle, nous essayons d’approximer la fonction de Friedman. Elle dépend uniquement des 5 premières variables d’entrée, ce qui en fait un bon cas d’étude pour la sélection de variables.

Soit un vecteur d’entrée $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$, avec $d \geq 5$. La fonction de Friedman #1 est définie par :

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon,$$

où $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ est un terme de bruit gaussien, ajouté pour simuler des observations bruitées.

Les variables x_i sont tirées uniformément sur l'intervalle $[0, 10]$.

Cette fonction présente plusieurs caractéristiques intéressantes : la présence de termes non linéaires et quadratiques. Voici des résultats obtenus sans bruit.

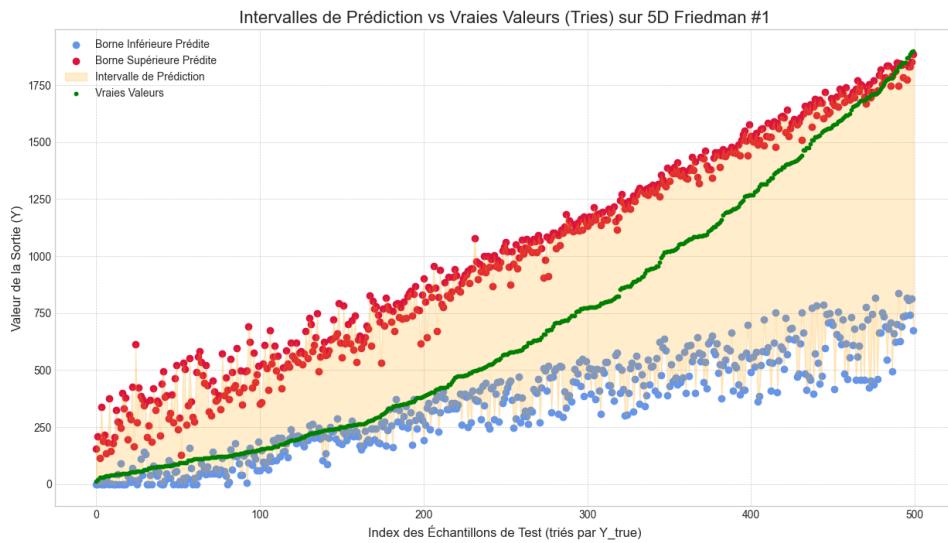


FIGURE 15 – Multi-layer Macsum avec fonction d’activation Leaky ReLU, 32 neurones en entrée, 16 en couche cachée, et 1 en sortie

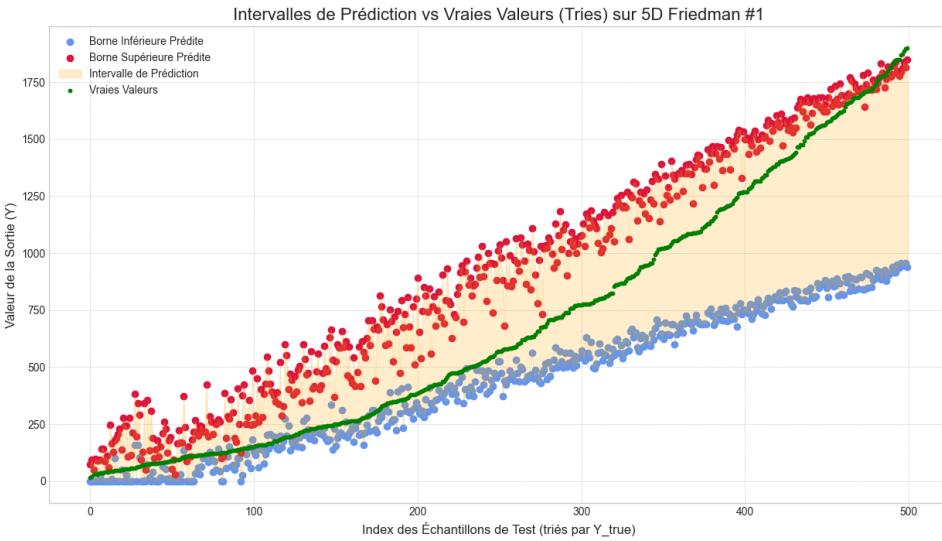


FIGURE 16 – Multi-layer Macsum avec fonction d’activation Leaky ReLU et une pénalité plus importante sur la taille de l’intervalle

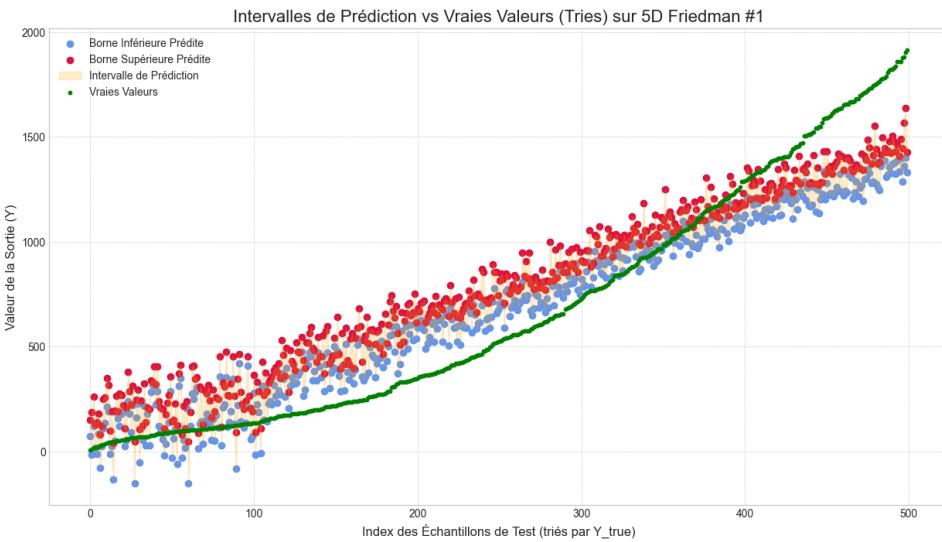


FIGURE 17 – Macsum avec SPIL

6 Hybride Multi-layer Macsum et Perceptron

6.1 Motivation et limites du modèle MacsumNet initial

Le modèle MacsumNet initial, bien que fonctionnel, présente des défis d'optimisation lorsque le réseau gagne en profondeur. La composition successive de couches Macsum crée un paysage de fonction de coût complexe. Pour surmonter ces difficultés et augmenter la capacité expressive du modèle, une architecture hybride a été conçue. L'objectif est de combiner la robustesse des couches de perceptron multicouches (MLP) pour l'extraction de caractéristiques avec la spécialisation de Macsum pour la prédiction d'intervalles.

6.2 Principe de l'arithmétique des intervalles

Pour construire un réseau hétérogène capable de propager des intervalles, il est nécessaire de définir comment les opérations standards (transformations linéaires, fonctions d'activation) affectent un intervalle. Ce cadre mathématique est fourni par l'arithmétique des intervalles.

Définition 1 : Opérateur d'intervalle Soit une fonction $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ et un hyper-rectangle (ou intervalle vectoriel) $[x] = [\underline{x}, \bar{x}]$ où $\underline{x} \leq \bar{x}$ (l'inégalité est comprise élément par élément). L'opérateur d'intervalle $[f]$ est une fonction qui à $[x]$ associe un nouvel intervalle $[y] = [f](x)$ tel que

$$\forall x \in [x], \quad f(x) \in [y].$$

Autrement dit,

$$\forall x \in [\underline{x}, \bar{x}], \quad f(x) \in [[f]\underline{x}, [f]\bar{x}].$$

L'objectif est de trouver l'opérateur $[f]$ qui produit l'intervalle le plus "serré" (le plus petit possible) tout en garantissant la propriété d'inclusion.

6.3 Propagation des intervalles dans les couches linéaires

Considérons une couche de perceptron standard, définie par la transformation affine $f(x) = Wx + b$. La propagation d'un intervalle à travers cette transformation est un résultat fondamental de l'arithmétique des intervalles.

Propriété fondamentale : transformation affine d'intervalle La propagation exacte (la plus serrée) d'un intervalle $[x] = [\underline{x}, \bar{x}]$ à travers la fonction $f(x) = Wx + b$ est donnée par les équations suivantes, comme décrit par Moore et al.

(2009) et utilisé dans les travaux sur la vérification des réseaux de neurones (Gowal et al., 2018) :

$$[y] = [\underline{y}, \bar{y}]$$

avec

$$\underline{y} = W^+ \underline{x} - W^- \bar{x} + b \quad (10)$$

$$\bar{y} = W^+ \bar{x} - W^- \underline{x} + b \quad (11)$$

où W^+ et W^- sont les parties positive et négative de la matrice de poids W , définies par :

$$W^+ = \max(0, W), \quad W^- = \max(0, -W).$$

6.4 Propagation à travers les fonctions d'activation

Pour les fonctions d'activation, la propagation dépend de leur monotonie.

Définition 2 : Propagation monotone Si une fonction d'activation $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ est monotone non décroissante (par exemple, ReLU, LeakyReLU, Sigmoid, tanh), sa propagation sur un intervalle $[a, b]$ est simplement :

$$\sigma([a, b]) = [\sigma(a), \sigma(b)] \quad (12)$$

Cette propriété simplifie grandement la construction du réseau, car elle permet d'appliquer l'activation directement et indépendamment sur chaque borne de l'intervalle.

6.5 Architecture du modèle hybride final

En s'appuyant sur ces principes, le modèle a été implémenté. Il est composé d'une séquence de couches hétérogènes :

- **Couches d'extraction (`IntervalLinearLayer`)** : Ces couches implémentent les équations (10), (11) et (12). Elles prennent un intervalle en entrée et produisent un nouvel intervalle représentant des caractéristiques de haut niveau. Leur rôle est de transformer les données brutes en une représentation sémantiquement plus riche.
- **Couches de prédiction (`MacsumLayer`)** : Ces couches opèrent également sur des intervalles. Elles appliquent l'opérateur Macsum (défini précédemment) sur les bornes inférieure et supérieure de l'intervalle d'entrée

de manière indépendante. La couche finale Macsum a pour rôle de réduire le vecteur de caractéristiques final à un intervalle de sortie scalaire $[\underline{y}_{\text{final}}, \bar{y}_{\text{final}}]$.

La fonction de coût globale du modèle reste SPIL, qui combine un terme de largeur d'intervalle

$$\alpha \times (\bar{y} - \underline{y})$$

et un terme de pénalité de non-contenance

$$\gamma \times \text{Penalty}(Y, \underline{y}, \bar{y}),$$

assurant ainsi que le modèle apprend à produire des intervalles à la fois précis et fiables.

Cette architecture synergique bénéficie de la stabilité et de la puissance d'extraction des MLP tout en conservant la capacité de Macsum à générer des prédictions d'intervalles interprétables.

7 Résultats Expérimentaux sur la Fonction de Friedman

Afin d'évaluer la performance et la stabilité des architectures hybrides, une série d'expériences a été menée sur la fonction de test synthétique de Friedman, en utilisant une version à 5 variables d'entrée (*features*). Pour toutes les expérimentations, les modèles ont été entraînés sur un jeu de données de 5000 échantillons pendant 2000 époques. Les hyperparamètres de la fonction de coût ont été fixés à $\alpha = 0.1$ et $\gamma = 0.8$ pour assurer un intervalle assez grand. L'objectif de cette section est d'analyser l'impact de deux choix architecturaux majeurs : le type de la couche d'entrée ('MacsumLayer'(M) vs. 'IntervalLinearLayer'(L)) et la composition des couches internes.

7.0.1 Analyse des Architectures avec une Couche d'Entrée Macsum

La première série d'expériences a porté sur des modèles initiés par une couche 'Macsum'. Cette approche postule que le mécanisme d'agrégation basé sur l'ordre de 'Macsum' peut être bénéfique dès le traitement des données brutes.

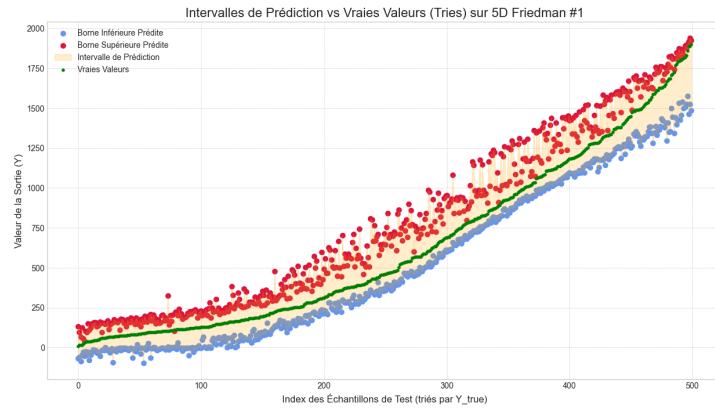


FIGURE 18 – Architecture 32M-16L-8L-8M-1M.

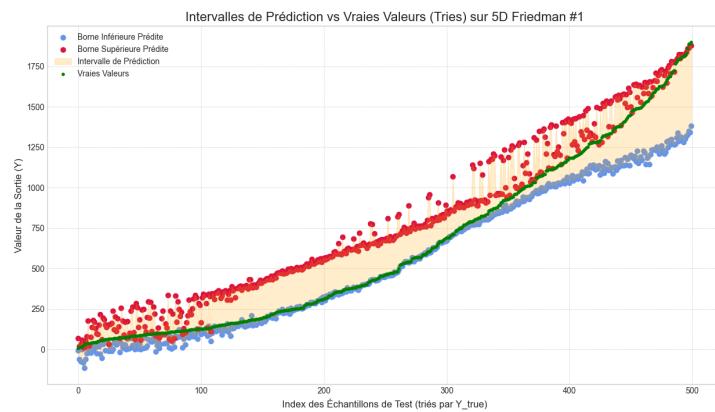


FIGURE 19 – Architecture 16M-8L-8M-1M.

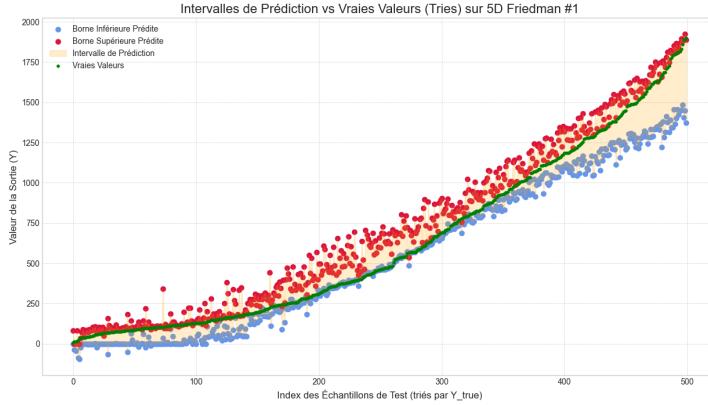


FIGURE 20 – Architecture 32M-8L-8M-4L-4M-1M.

Observation : Comme l’illustrent les figures 18 à 20, les architectures débutant par une couche ‘Macsum’ peuvent atteindre des résultats corrects, en particulier lorsque le réseau est suffisamment profond pour compenser les dynamiques d’optimisation complexes induites par cette première couche. L’un des avantages théoriques de cette approche est une potentielle meilleure interprétabilité et tolérance au donnée manquante, le paramètre φ de la première couche s’appliquant directement aux *features* d’entrée. Cependant, comme nous le verrons, cette configuration s’est avérée moins stable et moins performante que l’alternative.

7.0.2 Analyse des Architectures avec une Couche d’Entrée Perceptron

La seconde approche, qui s’est révélée plus fructueuse, consiste à utiliser une couche ‘IntervalLinearLayer’ (Perceptron) en entrée. La logique sous-jacente est de déléguer l’extraction de caractéristiques initiales à une couche ‘Linear’ robuste, avant de les passer à des modules ‘Macsum’ spécialisés.

Conception Architecturale : Les architectures testées suivent une philosophie de décomposition de la tâche :

1. **Extraction de Caractéristiques :** Une ou plusieurs couches ‘IntervalLinearLayer’ initiales.
2. **Agrégation et Raisonnement :** Une alternance de couches ‘Macsum’ et ‘Linear’ pour traiter et raffiner les caractéristiques de haut niveau.
3. **Prédiction Finale :** Une couche ‘Macsum’ terminale pour agréger l’information finale en un intervalle de prédiction.

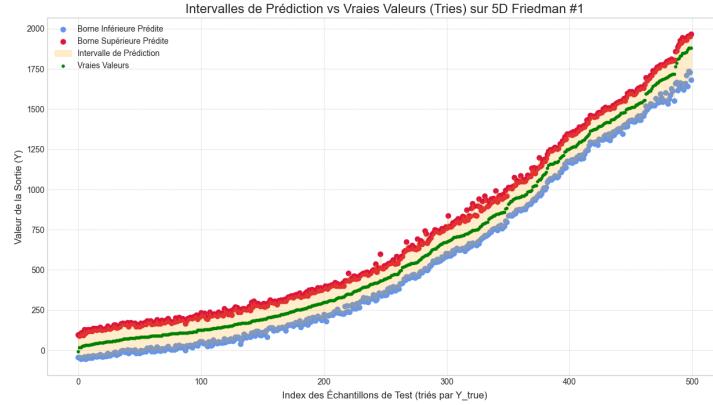


FIGURE 21 – Performance de l’architecture 16L-8M-4L-1M.

Analyse des performances : La figure 21 montre les résultats pour l’architecture ‘16L-8M-4L-1M’. On observe que, même avec un réseau de taille réduite par rapport aux expériences précédentes, le modèle produit des intervalles de prédiction lisses et de haute qualité. Il est à noter que la convergence n’était pas encore atteinte après 2000 époques, suggérant un potentiel d’amélioration supplémentaire avec un entraînement plus long, comme le confirmera la figure 26.

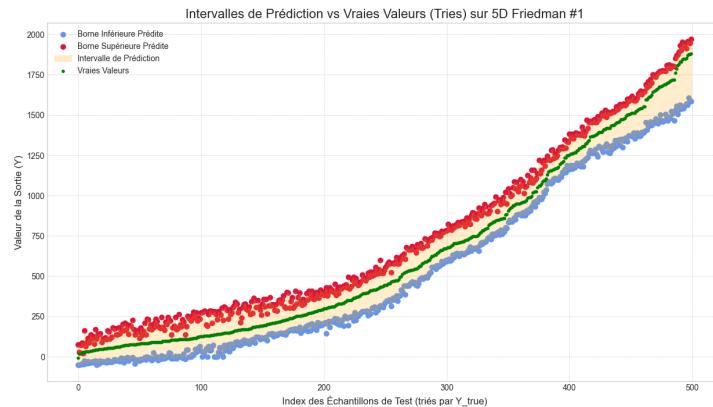


FIGURE 22 – Performance de l’architecture 8L-8M-4L-1M

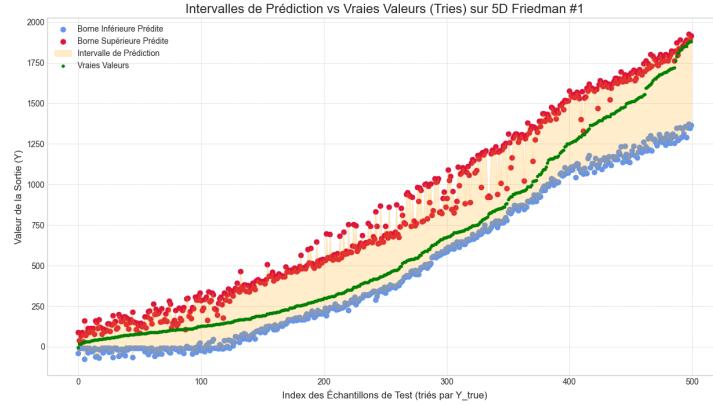


FIGURE 23 – Performance de l’architecture 8L-8M-1M

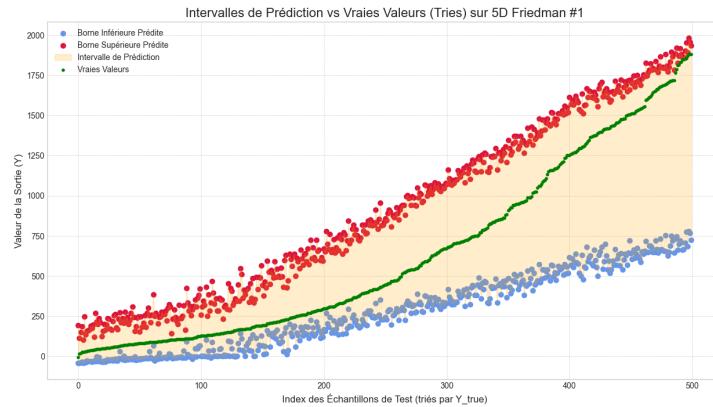


FIGURE 24 – Performance de l’architecture 8L-1M

Tests d’ablation et importance des couches : Pour comprendre le rôle de chaque composant, plusieurs tests d’ablation ont été effectués. En comparant diverses architectures, nous avons observé que la suppression de couches ‘Linear’ intermédiaires dégrade légèrement la précision, tandis que la suppression des couches ‘Macsum’ intermédiaires mène à une augmentation drastique et non maîtrisée de la largeur des intervalles. Ceci suggère que les modules ‘Macsum’ jouent un rôle actif et nécessaire dans l’agrégation et le contrôle de la propagation des intervalles.

Validation de l’approche hybride : L’expérience la plus concluante fut la comparaison avec un modèle purement ‘Perceptron’ (‘4L-2L-2L-1M’, où la dernière couche ‘Macsum’ est conservée pour la sortie).

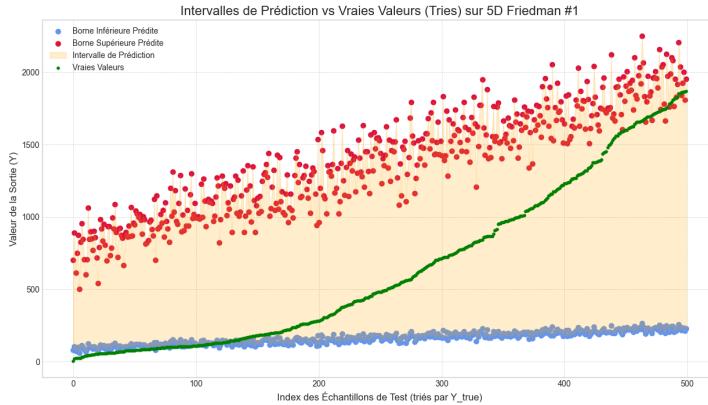


FIGURE 25 – Échec de convergence d'une architecture quasi-exclusivement Linear.

Comme le montre la figure 25, cette architecture échoue complètement. Ce résultat est fondamental : il prouve que les couches ‘Macsum’ ne sont pas interchangeables. Elles introduisent un **biais inductif** et une **régularisation structurelle** indispensables, en forçant le réseau à raisonner sur l'ordre des caractéristiques, un mécanisme que le MLP seul ne parvient pas à apprendre.

Performance des architectures compactes et effet de l'entraînement : Finalement, il a été démontré que des architectures compactes mais bien structurées, comme ‘4L-2M-2L-1M’ ou même ‘4L-1M-1L-1M’, sont capables de produire d'excellents intervalles. La figure 26 illustre que, avec un budget d'entraînement suffisant (4000 époques), le modèle ‘4L-2M-2L-1M’ présente à la fois une précision élevée et un intervalle de confiance resserré.

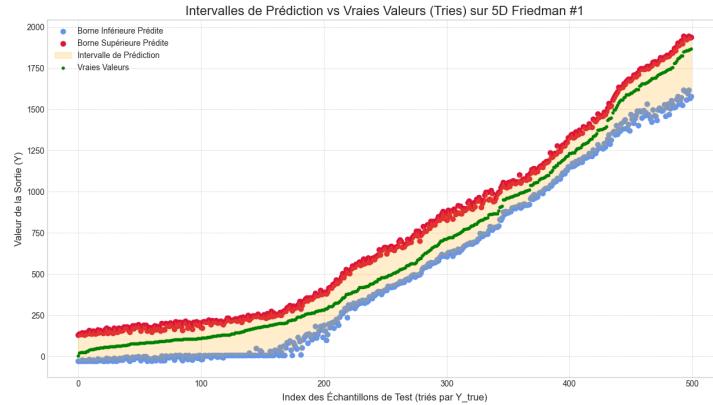


FIGURE 26 – Performance de l’architecture $4L-2M-2L-1M$ après 4000 époques.

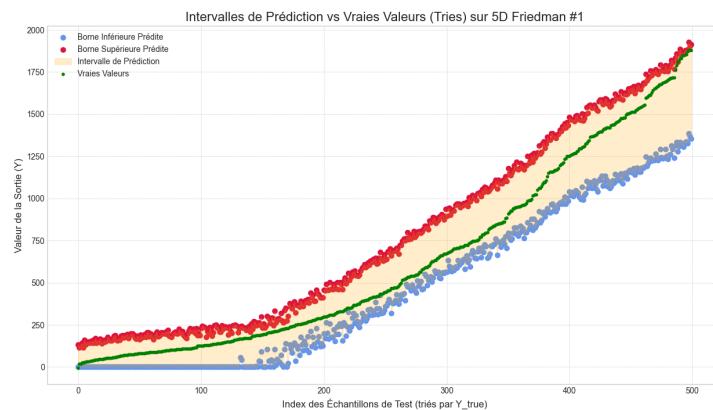


FIGURE 27 – Performance de l’architecture compacte et efficace $4L-1M-1L-1M$.

7.0.3 Synthèse des Résultats

Les expérimentations convergent vers une conclusion claire : les architectures hybrides débutant par des couches de type Perceptron sont nettement supérieures. Elles bénéficient de la stabilité des couches ‘Linear’ pour l’extraction de caractéristiques et de la puissance de ‘Macsum’ pour l’agrégation non-linéaire et la prédiction d’intervalles. Le rôle des couches ‘Macsum’ s’est avéré non seulement bénéfique mais nécessaire, agissant comme un régulateur structurel qui guide l’optimisation vers des solutions performantes et robustes.

8 Conclusion

L’approche par l’Interval Mean Squared Error (IMSE) est plus pertinente lorsque l’on dispose d’une grande puissance de calcul (et donc d’un grand volume de données), et dans des espaces de faible dimension. En revanche, dans des espaces de grande dimension, voire avec peu de données, la Sigmoid Penalized Interval Loss (SPIL) permet d’entraîner le modèle avec d’excellentes performances, tout en nécessitant une puissance de calcul bien moindre que l’IMSE.

Par ailleurs, dans notre cas, les paramètres de SPIL ont été utilisés de manière générique. Une optimisation fine de ces paramètres pourrait permettre de réduire la taille d’une borne de l’intervalle de prédiction, ou d’améliorer la précision du modèle. Il serait également possible, en séparant l’hyperparamètre γ en deux hyperparamètres distincts, de moduler l’importance des erreurs selon qu’elles concernent la borne inférieure ou la borne supérieure.

SPIL semble donc très modulable, mais à ce jour, aucune expérimentation n’a encore été menée pour évaluer l’influence spécifique de chacun de ses hyperparamètres.

L’exploration des architectures Macsum Network a mené au développement d’un modèle hybride, intégrant des couches de type Perceptron (Linear) aux côtés des couches Macsum. Cette nouvelle approche a été motivée par la nécessité de surmonter les défis d’optimisation rencontrés avec les réseaux Macsum profonds, tout en capitalisant sur leurs propriétés uniques.

Les résultats obtenus avec le modèle Hybrid Network se sont montrés très prometteurs. Il a été observé qu’une architecture débutant par des couches Linear pour l’extraction de caractéristiques, suivies de couches Macsum pour l’agrégation et la prédiction, offre un bien meilleur compromis entre stabilité et performance. Expérimentalement, cette configuration a permis de former des réseaux plus profonds et de produire des intervalles de prédiction de haute qualité, à la fois fiables et précis.

Un résultat particulièrement notable est que les couches Macsum ne sont pas de simples composants interchangeables. Les tests ont montré qu’un réseau purement Linear de structure équivalente ne parvenait pas à converger correctement, ce qui souligne le rôle essentiel du mécanisme de Macsum dans la réussite du modèle.

En conclusion, bien que cette étude constitue une première approche, elle valide l’intérêt de combiner différentes philosophies de couches neuronales. Le modèle HybridNet représente une base de travail solide, démontrant qu’il est possible d’atteindre une excellente précision tout en conservant un potentiel d’interprétabilité grâce aux couches Macsum

9 Références bibliographiques

- Gowal, S., Dvijotham, K., Stanforth, R., et al. (2018). On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. arXiv :1810.12715.
- Moore, R. E. (1966). Interval Analysis. Prentice-Hall. *Ouvrage fondateur du domaine de l'analyse par intervalles*.
- Moore, R. E., Kearfott, R. B., Cloud, M. J. (2009). Introduction to Interval Analysis. Society for Industrial and Applied Mathematics (SIAM).
- Hmidy, Y., Rico, A., Strauss, O. Macsum Aggregation Learning.

10 Annexe

- α : 0.1
- γ : 0.8
- $k_{sigmoid}$: 0.1
- Taille du kernel : 2

Résultats agrégés de la validation croisée

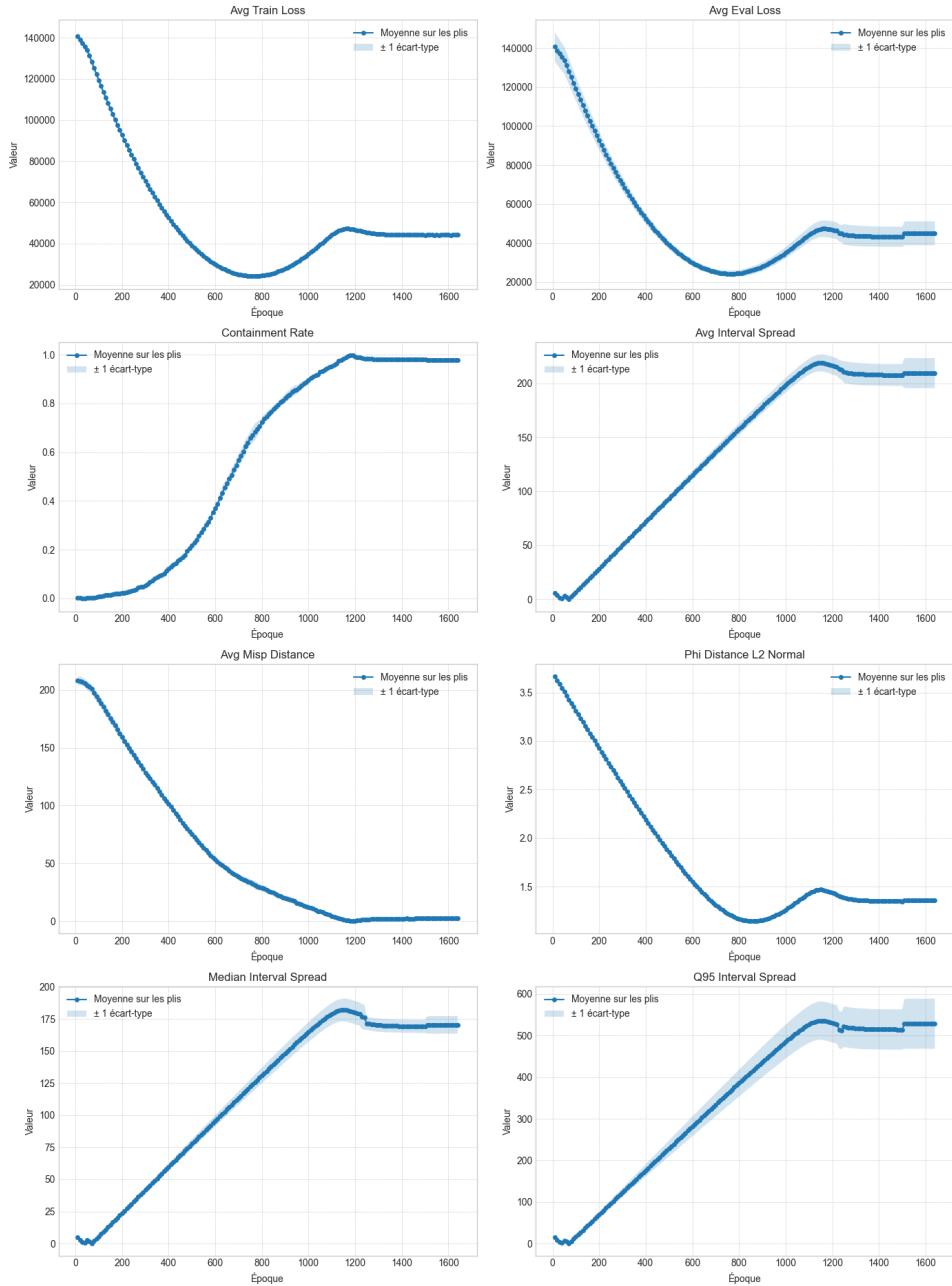


FIGURE 28 – Loss : IMSE

Résultats agrégés de la validation croisée

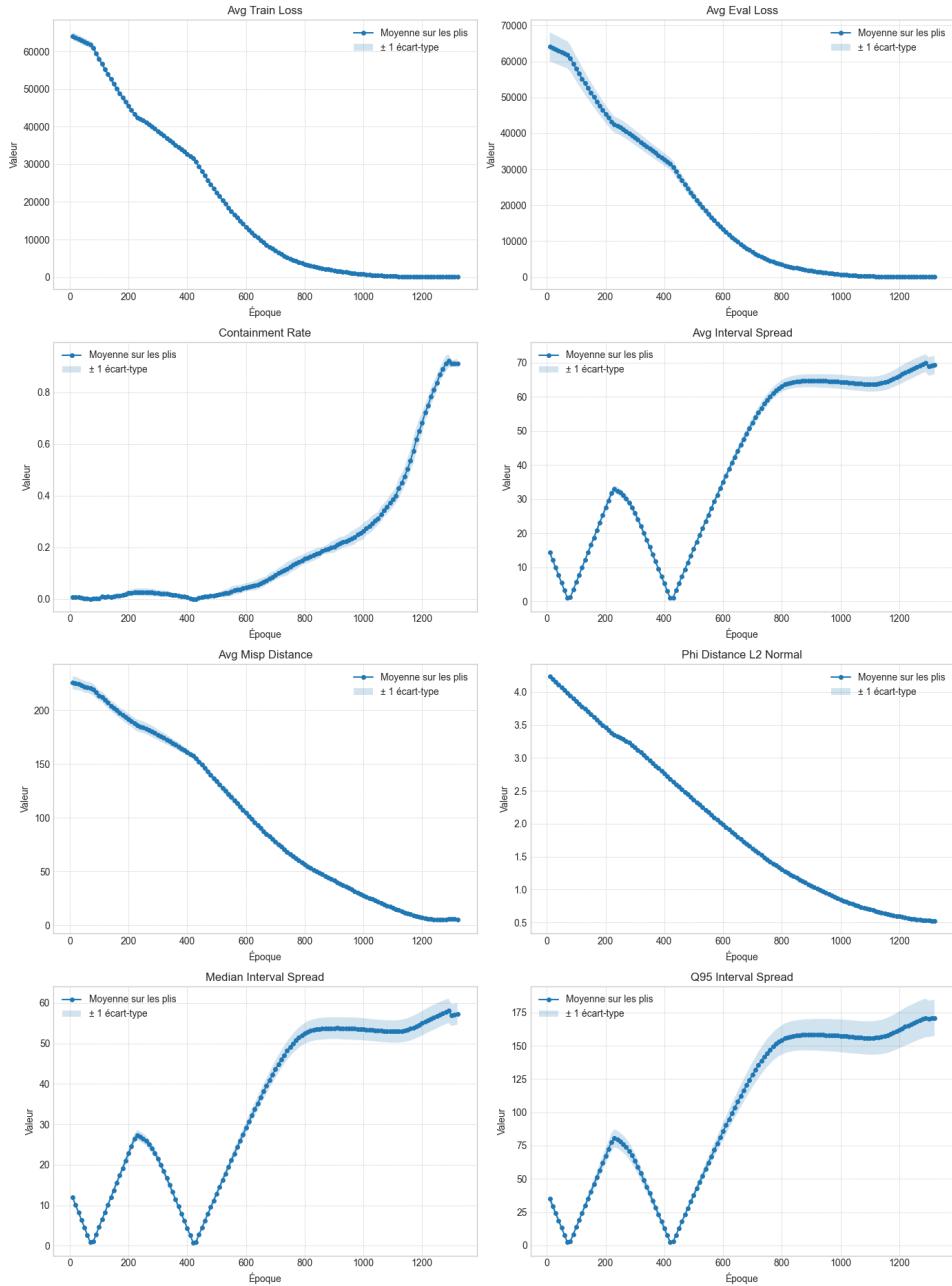


FIGURE 29 – Loss : SPIL

- α : 0.1
- γ : 0.8
- $k_{sigmoid}$: 0.1
- Taille du kernel : 20

Résultats agrégés de la validation croisée

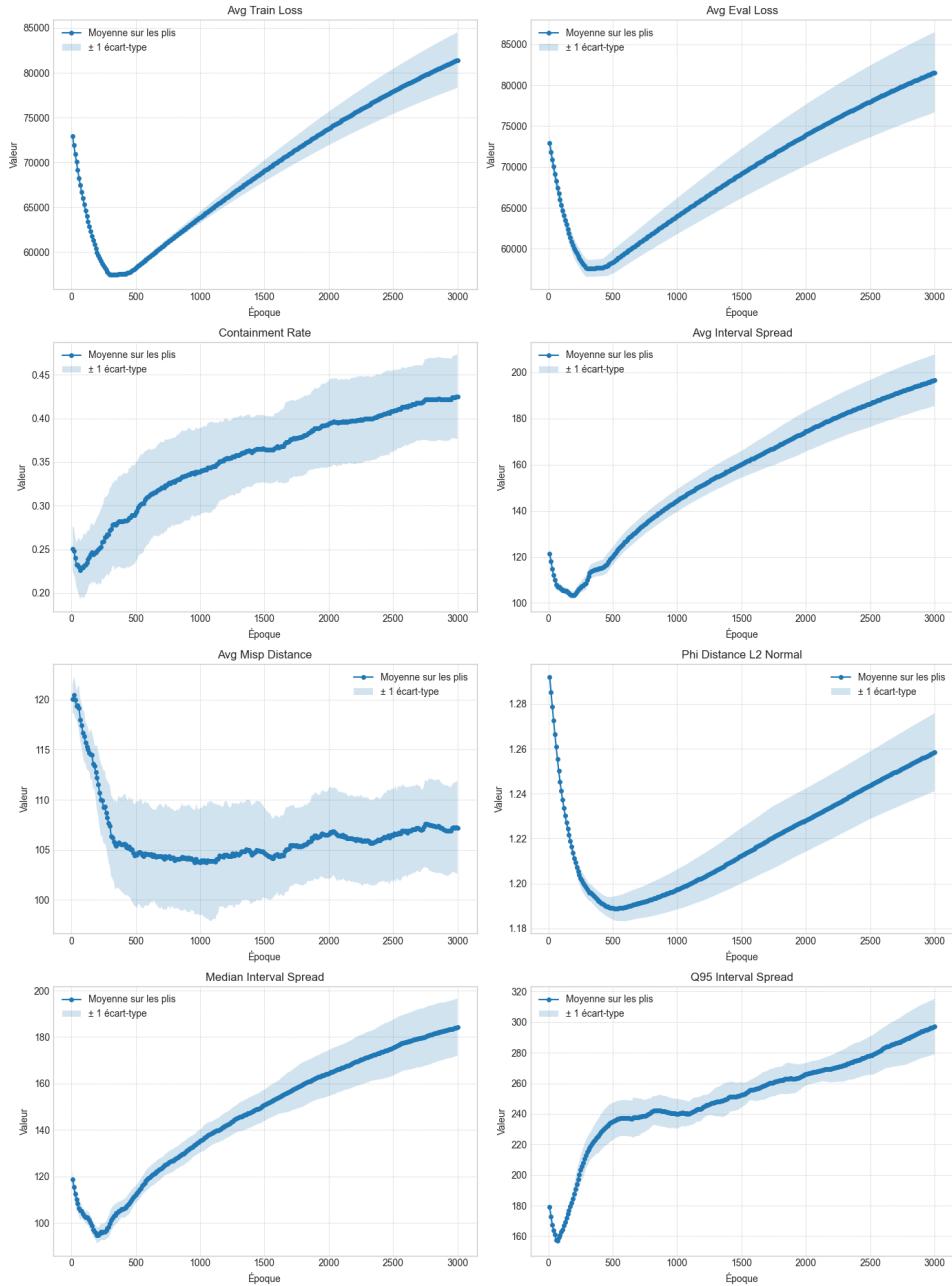


FIGURE 30 – Loss : IMSE

Résultats agrégés de la validation croisée

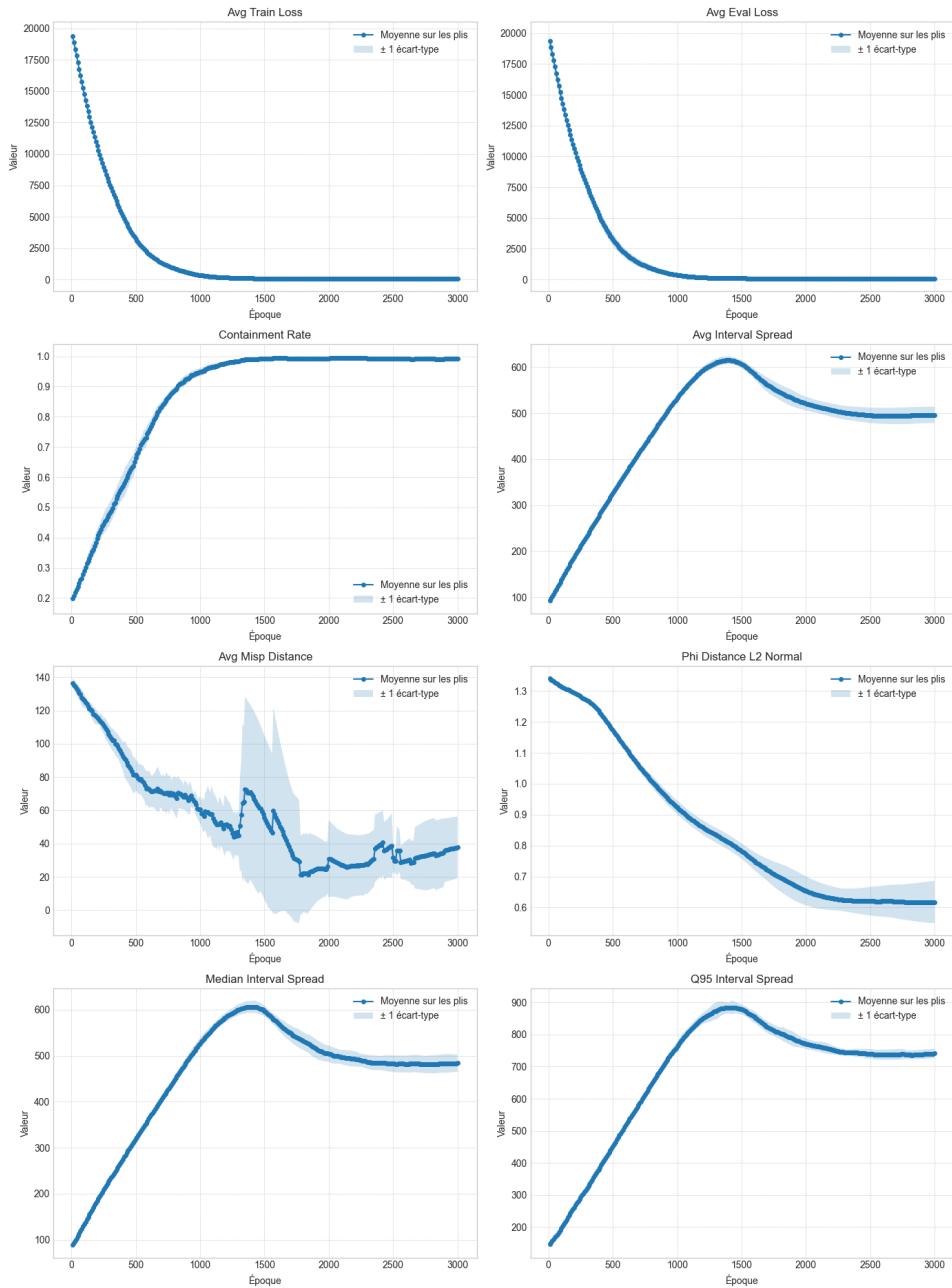


FIGURE 31 – Loss : SPIL

- α : 0.1
- γ : 0.8
- $k_{sigmoid}$: 0.1
- Taille du kernel : 100

Résultats agrégés de la validation croisée

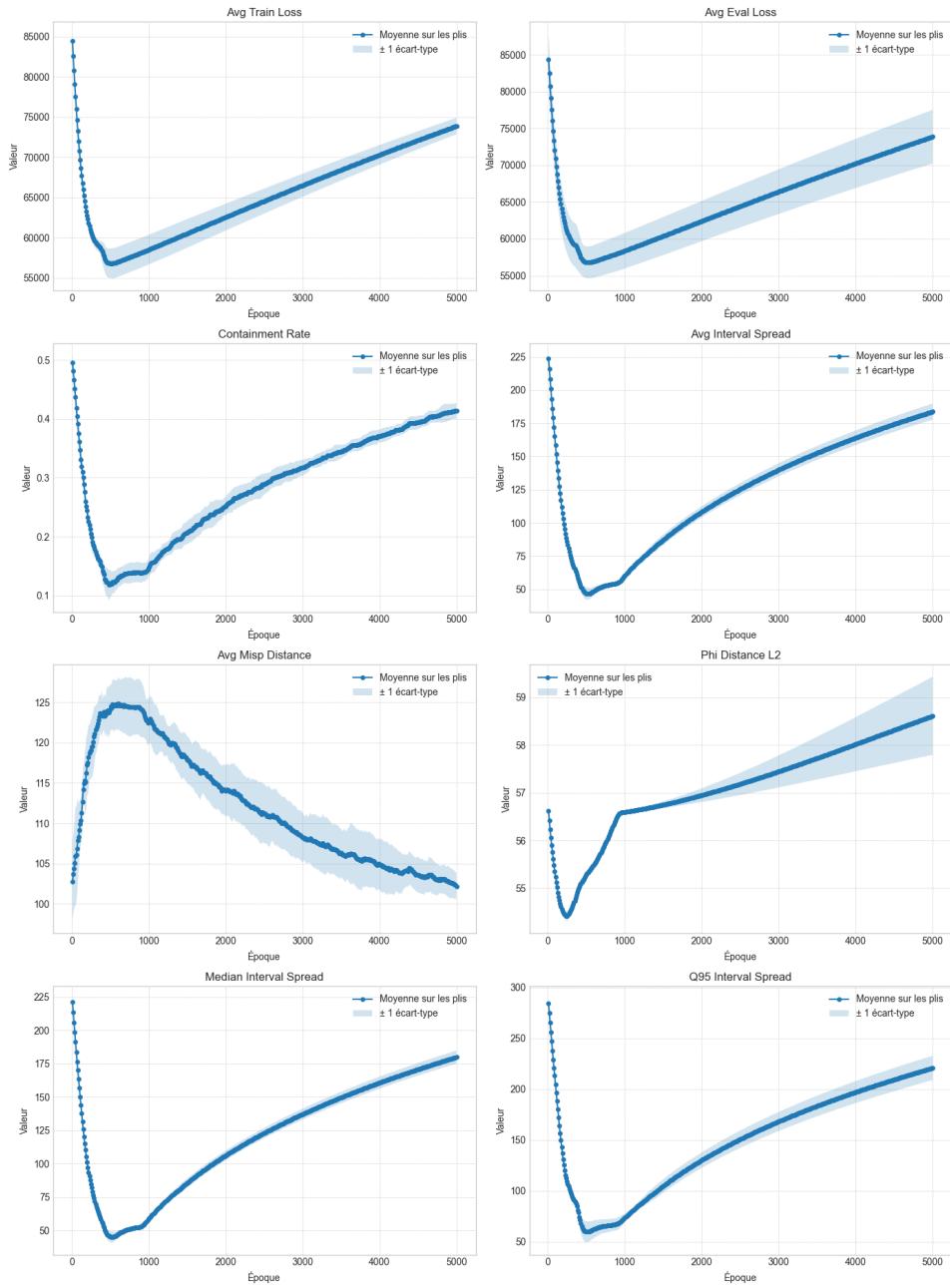


FIGURE 32 – Loss : IMSE

Résultats agrégés de la validation croisée

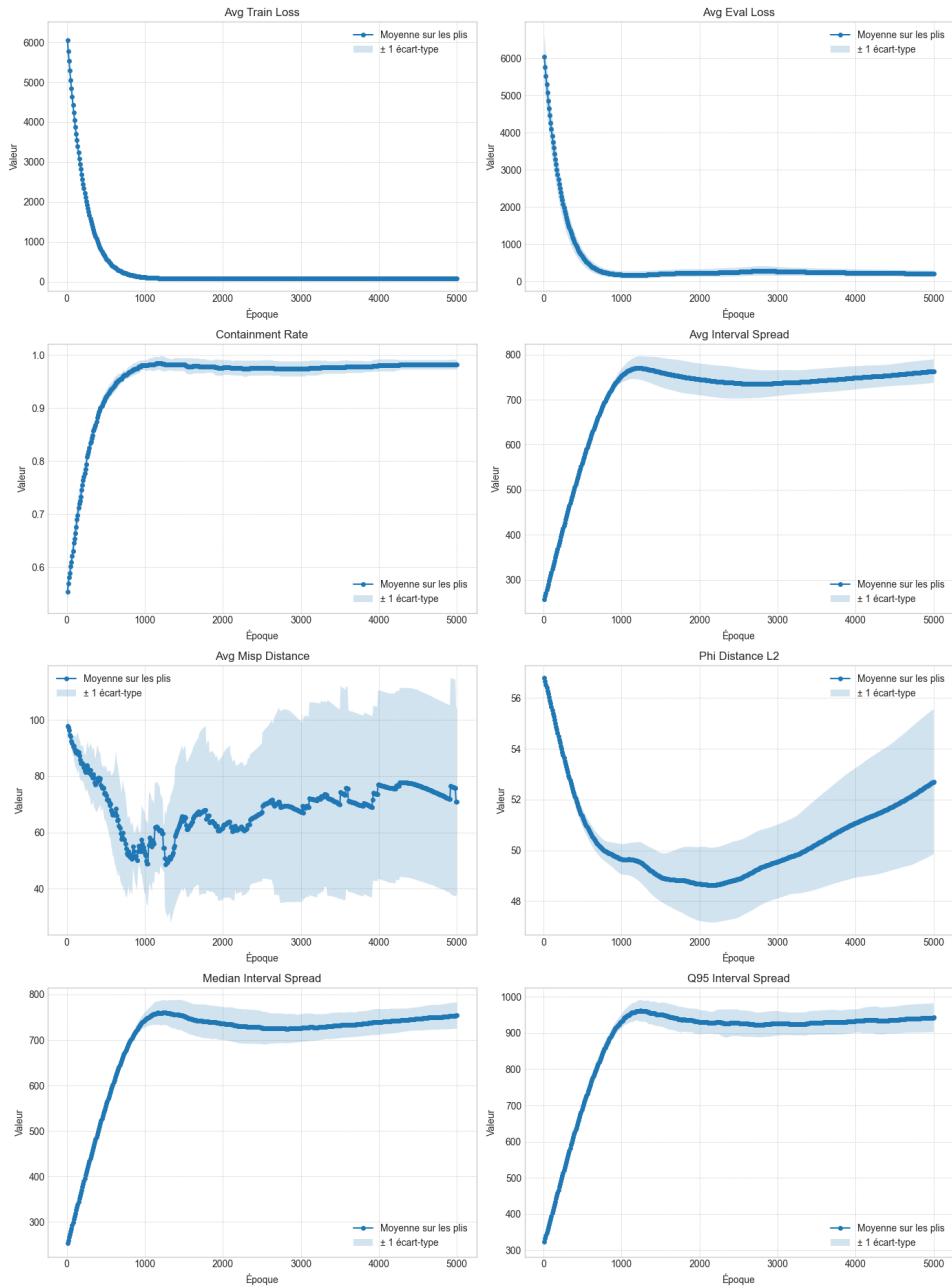


FIGURE 33 – Loss : SPIL