

La convolution lissée ajustable

Couche d'entrée de réseau de neurones pour l'analyse
de données fonctionnelles

Valentin Larchevêque

IMAG

Faculté des sciences de Montpellier
France



STATISTIQUE
SCIENCE DES DONNÉES
UNIVERSITÉ DE MONTPELLIER



Professeur Cédric Beaulac

STATQAM

Université du Québec À Montréal
Canada



01/04/2023

→

31/08/2023

TABLE DES MATIÈRES

1	<u>Introduction :</u>	4
1.1	Problématique	4
1.2	Littérature et état de l'art	6
1.3	Contribution et résultats	7
1.4	Le plan	8
2	Outils préalables	9
1	L'analyse de données fonctionnelles	9
1.1	Introduction à l'analyse de données fonctionnelles :	9
1.2	Représentation	12
1.3	Dérivées :	15
2	L'apprentissage automatique	16
2.1	Qu'est-ce qu'un modèle d'apprentissage	16
2.2	Décomposition biais-complexité	18
3	Le réseau de neurones artificiel	19
3.1	Définition d'un réseau de neurones artificiel	19
3.2	Un estimateur de fonction universel	21
3	L'opération de convolution	23
3.1	Convolution et probabilités	23
3.2	La couche de convolution	24
3.3	Propriétés de la convolution	26
3	<u>Convolution lissée ajustable</u>	29
1	Convolution sur données fonctionnelles :	29
1.1	Problèmes de ce modèle :	30
2	La convolution lissée ajustable : Description du principe	32

2.1	Avantages de la convolution lissée :	33
2.2	Une généralisation de la convolution :	34
2.3	Continuum entre convolution discrète et continue	37
4	Expérimentations	41
2	Méthode d'expérimentation	41
2.1	Jeux de données	41
2.2	Évaluation des performances :	42
2	Choix de l'architecture :	42
2.1	Nombre de couches :	43
2.2	Couches non-paramétriques :	44
2.3	Choix du regroupement (Pooling) :	47
2.4	Paramètres propres à TSC :	48
3	Comparaison à d'autres modèles	53
3.1	Convolution discrète classique vs TSC	53
3.2	Comparaison générale	55
3.3	Un exemple de régression :	58
	<u>Conclusion :</u>	59
	<u>Remerciements :</u>	62
	ANNEXE	63

Résumé :

Imaginez devoir analyser la croissance des enfants au cours du temps. Prendre les mesures toutes les secondes serait très coûteux pour les expérimentateurs et pour les enfants, sachant de plus que la taille d'un enfant mesurée une seconde après la dernière prise de mesure risque de ne pas apporter de grandes variations. C'est pourquoi dans le cadre d'études longitudinales il est coutume de prendre les mesures à des temps relativement espacés selon les besoins et les expériences. Cependant, sur un grand nombre d'enfants ou même de patients, il paraît impraticable de mesurer les variables d'intérêt, comme la taille ou la teneur en oxygène du sang par exemple, aux mêmes temps de leurs vies, le même jour, même heure, toutes mesurées sur une seule et unique grille d'échantillonnage, comportant le même nombre de mesures pour tous les individus. Il est également impraticable, ou parfois inutile de mesurer de manière régulière les patients, qui ont besoin de plus d'attention à certains moments de leur vie qu'à d'autres moments. On se retrouve donc à devoir faire des choix pratiques d'échantillonnage ce qui donne lieu à un type de données bien particulières : les données fonctionnelles. Elles représentent des défis pour les statisticiens cherchant à guider la prise de décisions telles que les formulations de diagnostique ou les prises en charges médicales par exemple. Dans ce travail nous nous intéressons à la régression de fonction sur scalaire, *Function on scalar* (FoS) et cherchons à prédire des nombres ou des classes à partir de données structurées de la sorte.

Les réseaux de neurones artificiels sont de puissants modèles de prédiction qui ont fait leurs preuves avec les avancées récentes surtout en matière d'analyse d'image, ou encore de traitement du langage naturel avec le désormais célèbre ChatGPT d'openAI. Ces outils ont des avantages considérables en terme de performance mais leur utilisation sur des données structurées de façon fonctionnelle n'est pas évidente, notamment à cause du nombre de données disponible qui peut varier d'un patient à l'autre, le phénomène de censure étant très courant dans les essais cliniques.

Dans ce travail, nous poursuivons les travaux qui ont été effectués pour introduire les réseaux de neurones comme modèles d'analyse de données fonctionnelles, en apportant une nouvelle classe de modèles, basés sur la convolution comme en analyse d'image, ainsi que sur des lissages classiques de l'analyse de données fonctionnelles.

Notre contribution n'est pas l'amélioration des performances de l'état de l'art actuel mais dans l'avènement d'une toute nouvelle classe de modèles placés sur un continuum entre d'une part l'analyse "d'image" unidimensionnelles que sont ces données fonctionnelles et d'autre part le processus de filtrage en théorie du signal, et qui répondent aux enjeux majeurs de l'analyse de données fonctionnelles.

CHAPITRE 1

INTRODUCTION:

1.1 Problématique

L'analyse de ces données fonctionnelles (*Functional Data Analysis* ou FDA) est un domaine important des statistiques modernes. Là où en analyse multivariée classique on observe des variables d'individus, en analyse de données fonctionnelles on évalue une ou plusieurs variables en fonction d'une ou plusieurs autres en différents temps. Là où par exemple la taille d'un individu pris au hasard dans une population va suivre une distribution dont on va chercher à estimer les paramètres, la taille de cet individu en fonction du temps semble être un processus continu dont, lorsque l'on en fait la mesure, est échantillonné à un temps précis. Puisqu'il serait absurde de considérer que cet individu ne grandit qu'au moment de la prise de sa mesure, on s'intéresse alors à sa courbe de croissance que l'on imagine lisse, lui étant propre et que l'on suppose être la réalisation d'une variable aléatoire fonctionnelle, dont on peut essayer d'estimer la forme plus générale à partir d'observations d'autres individus, à des temps précis. On peut également vouloir classifier l'individu selon son sexe grâce à sa courbe de croissance, les courbes de croissances des hommes et des femmes n'ayant pas forcément la même forme, ou bien plus intéressant comme exemple, pré-diagnostiquer une maladie à partir de données médicales comme les courbes de poids, de concentrations sanguines diverses ou encore d'ondes cérébrales. Bien que les outils communs de l'analyse multivariée ponctuelle et de la statistique inférentielle que sont les fonctions de densité et l'estimation paramétrique soient inutilisables dans ce type d'analyse, d'autres outils comme les expansions de bases et les dérivées s'ouvrent au statisticien pour l'analyse de ce type de données.

Puisque les observations sont supposées être issues de processus lisses, un enjeu considérable de l'analyse de données fonctionnelles est la représentation machine de ces objets. En effet puisque les fonctions sont des objets de dimension

infinie, alors il est impossible de les stocker complètement. Pour remédier à ce problème et réduire la dimension des objets desquels on traite, un des pré-traitement courant est l'expansion de base de Schauder, qui sont certes des bases infinie, mais le principe d'expansion est de tronquer la base pour approcher arbitrairement bien les fonctions. La méthode consiste à exprimer les variables fonctionnelles comme combinaison linéaire d'un nombre finit de coefficients avec des fonctions d'une base. La base que nous utilisons le plus dans ce travail est la base de B-spline qui consiste à mettre bout-à-bout des fonctions polynomiales en les connectant en des noeuds et manière lisse. Cette approche est considérée par J.O Ramsay et Silvermann [6] comme la meilleur pour les données ne présentant pas de saisonnalité car elle coûte aussi peu en calcul que les lissages polynômiaux mais ont l'avantage d'être bien plus précis et maniables de part le fait que l'on puisse choisir exactement la place des noeuds sur les supports des observations. Ainsi des fonctions ayant énormément de bruit ou même de variations dans les premières mesures et assez peu de variation vers la fin de son support sont bien approchables par une expansion de base de B-spline car on peut ajouter plus de noeuds au départ ce qui permet d'avoir une bonne représentation des variations du début sans altérer la représentation de la queue de la fonction. La base de Fourier est une autre base de Schauder permettant de faire des expansions mais est surtout efficace pour représenter des fonctions présentant une saisonnalité, en plus d'être moins flexibles car ne permettant pas à un système de noeuds de paramétriser à souhait les fonctions obtenues. C'est pour ces raisons que nous n'utiliserons que les bases de B-splines bien qu'il y a d'autres choix pour représenter les données fonctionnelles soient envisageables.

Dans ce projet, nous nous sommes concentrés essentiellement sur la tâche de classification fonctionnelle, qui consiste à étiqueter de manière automatique des données fonctionnelles selon leurs classes. De nombreuses classifieurs existent pour ce problème comme les algorithmes de plus proches voisins [5] ou les régressions logistique pour la classification binaire par exemple, cependant mon but a été au cours de ce projet, en tant que stagiaire de recherche, de participer au vaste projet du professeur Cédric Beaulac, professeur à l'Université du Québec à Montréal, d'intégrer les outils d'apprentissage automatique et d'apprentissage profond à l'analyse de données fonctionnelle.

Les réseaux de neurones artificiels sont en effet, puissants outils pour l'analyse et la prédiction dans divers domaines, et ont fait leurs preuves au cours des dernières années dans plusieurs tâches, qu'il s'agisse de problèmes statistiques, de classification d'image et même de génération de texte jusque très récemment. Il sont en réalité des estimateurs de fonctions universelle, comme le prouve le théorème d'approximation universelle démontré par Shai Ben-David et Shai Shalev-Schwartz [9]. Il est alors naturel d'imaginer leur utilisation dans l'analyse de données fonctionnelles. Cependant, plusieurs problèmes font leur apparition lors ce que l'on souhaite faire apprendre une tâche à un tel modèle sur des données fonctionnelles. En effet les données fonctionnelles vues comme des vecteurs d'une seule dimension peuvent ne pas être de la même taille car les observations ne

sont pas toutes prises aux même endroits sur la grille, et certaines observations peuvent être mesurées moins de fois que d'autres, ou à des temps différents de la grille de prise de mesures. Cela pose problème pour étant donnée l'architecture des réseaux de neurones complètement connectés, dont la dimension de la couche d'entrée doit être fixée à l'avance, car on ne peut alors pas lui donner des observations de taille différentes. Et même si les observations comportent tous le même nombre de prises de mesures mais à des temps différents, les réseaux de neurones complètement connecté traitera ces mesures de manière similaire, et ce en ignorant complètement à la fois la structure sous-jacente des observations et les hypothèses fondamentales de l'analyse de données fonctionnelles.

La problématique à laquelle nous avons tenté de répondre dans ce projet est de fabriquer une couche d'entrée de réseau de neurones pour les données fonctionnelles. Cela vient avec les problèmes évoqués avant c'est à dire l'irrégularité dans la prise de mesure, le fait de traiter des objets de dimensions infinie dans des tenseurs informatiques de dimension finie, le fait de prendre en compte les hypothèse de structures lisses des processus sous-jacents aux observations, et la corrélation forte des mesures qui sont proches.

1.2 Littérature et état de l'art

Les solutions actuelles d'apprentissages profonds pour l'analyse de données fonctionnelles ont surtout consisté à apprendre soit des données brutes avec des modèles de réseau de neurones artificiels denses (*Multilayer-Perceptron* ou MLP) comme l'ont fait Barinder Thind *et al* [13], essentiellement sur des données fonctionnelles régulièrement mesurées et espacées. Une de leurs contributions a été d'ajouter des covariables scalaires à l'entrée du réseau de neurones.

D'autres chercheurs, notamment H.Wang *et al.* [14] ont introduit des modèles de réseaux de neurones pour la régression de fonction sur scalaire, en utilisant des réseaux de neurones récurrents (*Recurrent Neural Networks* ou RNN) et même des réseaux de neurones convolutifs (*Convolutional Neural Networks* ou CNN) [14]. Les chercheurs F.Rossi *et al.* [7] ont utilisé diverses représentations (bases de Fourier, B-Spline et décomposition en composantes fonctionnelles principales) avec échantillonnages arbitraire de ces représentations qu'ils entrent dans un réseau de neurones dense.

Dans un autre sens, le professeur Cédric Beaulac et Sidi Wu ont introduit des modèles de régression de scalaire sur fonction (SoF) [15]. Dans ce projet, pour prédire des fonctions à partir de scalaire, les deux chercheurs proposent d'apprendre les coefficients de bases de B-splines, qui ont l'avantage d'être en nombre finit et donner des courbes lisses en sortie. La prise en compte de l'hypothèse de régularité des fonctions dans les réseaux de neurones est une des améliorations sur lesquelles le professeur C.Beaulac se concentre dans son projet général d'introduction des outils d'apprentissage profond pour la FDA. Nous nous inscrivons avec ce projet dans la continuité de ces travaux en ajoutant la couche de convolution

comme option aux statisticiens et praticiens souhaitant analyser des données fonctionnelles, et notre pré-traitement des données sera essentiellement de lisser par expansion de B-splines. L'idée du professeur Cédric Beaulac a été la suivante : les mesures d'une données fonctionnelles sont d'autant plus corrélées qu'elle sont proches en temps. Pour en revenir à l'exemple du résumé, les tailles d'un enfant mesurées à deux secondes d'écart vont être fortement proche. C'est pour cela qu'appliquer un réseau de neurone dense sur toutes les observations comme l'ont fait d'autres chercheurs comme des vecteurs d'une dimension avec toutes les mesures ensemble semble moins prendre en compte ce fait.

Concernant les données irrégulièrement mesurées, F.Rossi *et al.* [7] proposent d'évaluer les observations lissées par diverses bases en une grille régulière avant de les rentrer dans un réseau dense. Leur solution est efficace et défini l'état de l'art actuel pour régler le problème de l'espacement des mesures.

1.3 Contribution et résultats

Nous cherchons à appliquer le réseau de neurones artificiel sur les données fonctionnelles plus généralement, en incluant non seulement les données irrégulièrement espacées, mais en aussi en incorporant les hypothèses et outils classiques de la FDA comme les lissages par B-spline et leurs dérivées, pour tirer un maximum de la structure fonctionnelle et supposée lisse bien particulière de ce type de données, au lieu de les traiter comme des vecteurs aléatoires discrets sans tenir compte de leur nature.

La solution que nous proposons est de combiner les outils d'analyse de données fonctionnelles et l'opération de convolution qui a fait ses preuves récemment en tant que modèle de classification d'images [1]. Le modèle que nous proposons passe outre les problèmes que rencontrent les autres réseaux de neurones quand il s'agit de l'analyse de données irrégulièrement espacées dans les mesures, ainsi que l'intégration des hypothèses fondamentales de la FDA. Nous proposons un modèle à mi-chemin entre les modèles de convolution unidimensionnelle et un modèle de convolution continue, dont les paramètres offrent un vaste choix d'architecture et une flexibilité incomparable. Il s'agit d'une généralisation de la convolution discrète dans le cadre de la FDA, qui permet en plus d'intégrer les dérivées des données fonctionnelles comme étant des canaux de la couche de convolution. Les fondements théoriques de ce modèle sont prometteurs ainsi que ses performances en pratique, mais c'est surtout sa flexibilité dans le choix des paramètres qui peut rendre le modèle efficace étant donné la flexibilité des paramètres à choisir manuellement. Notre modèle consiste dans la première couche à lisser les observations avec le outils classique de l'analyse de données fonctionnelles que sont les expansions de bases (surtout les B-spline), et d'appliquer une couche de convolution très flexible à ce vecteur lissé. Le lissage pouvant être évalué en un nombre arbitrairement grand de points, les paramètres de la couche de convolution d'entrée peuvent être choisis d'une multitude de façon, et

la dilatation a particulièrement un rôle théorique de contrôle de la complexité très intéressant. Ce modèle est plus général que le MLP et que la convolution classique, souvent plus performant empiriquement que ces derniers et que les RNN sur les jeux de données utilisés pour les comparaisons, bien que pouvant être coûteux en calcul selon les paramètres. On a également accès aux sorties des couches de convolution qui peuvent permettre une visualisation du procédé d'apprentissage.

1.4 Le plan

Avant d'introduire notre modèle, nous allons commencer par introduire la théorie qui supporte notre modèle. Nous commencerons tout d'abord par introduire les outils classiques de l'analyse de données fonctionnelles, à savoir les hypothèses, les méthodes d'expansion de bases pour le lissage des données. Nous évoquerons également l'importance des dérivées dans ce type d'analyse ainsi que les justifications qui nous ont poussé à les incorporer dans notre modèle. Par la suite nous établirons le formalisme de l'apprentissage automatique tel qu'il est développé par Shai Ben-David et Shai Shalev-Schwartz [9], ce qui nous permettra d'établir la notion de classe d'hypothèse \mathcal{H} ainsi que le compromis biais-complexité, qui furent des guides de nos idées et hypothèses au cours de ce travail. C'est dans cette partie que nous formaliserons le réseau de neurones comme classe de modèle d'apprentissage, les avantages et garanties théoriques de ceux-ci. Ensuite, avant d'expliquer notre modèle et son principe, nous introduirons la notion de convolution d'un point de vue probabiliste et feront le lien avec la convolution tel qu'utilisée dans le cadre de l'apprentissage automatique. Ensuite nous exposerons le modèle que nous proposons, ses avantages et garanties théoriques avant de le comparer avec des modèles existants dans le domaine au cours d'expérimentations sur 4 jeux de données de classification fonctionnelle avec chacun leurs défis à relever. Enfin nous comparerons notre modèle aux autres sur le jeu de données de régression Tecator souvent utilisé dans l'état de l'art pour comparer les performances des modèles à travers la littérature scientifique [14],[7].

CHAPITRE 2

OUTILS PRÉALABLES

1 L'analyse de données fonctionnelles

1.1 Introduction à l'analyse de données fonctionnelles :

En statistique multivariée classique, on s'intéresse à l'analyse d'un nombre finit de variables mesurées sur des individus, et on voit ces objets comme des réalisations d'un vecteur aléatoire dont l'estimation de la distribution a souvent un rôle central dans son analyse. En analyse de données fonctionnelles, on s'intéresse à un ensemble d'observations de valeurs en fonction d'une ou plusieurs autres variables. On suppose en général que ces observations sont issus de mesures à des temps discrets d'un processus aléatoire, supposé continue et "raisonnablement" lisse. Chaque point de la fonction peut être vu comme une variable, et on considère alors que la dimension de ces objets est infinie, et si ils sont continus alors les voisinages de chaque point de ces objets sont très corrélés. L'ordre des variables, c'est-à-dire des valeurs mesurées de ces observations, ne peut pas être altérée sous peine de modifier la structure de ces-dernières, contrairement à l'analyse multivariée classique, où l'ordre des variables n'a pas d'importance. Les fonctions sont alors des réalisations de processus stochastiques continus appelée variable aléatoire fonctionnelles mais dont les observations sont collectées à des temps discrets.

Commençons d'abord par définir formellement ce que l'on entend par "donnée fonctionnelle". Introduisons désormais un exemple en détail d'une donnée fonctionnelle ou d'une observation fonctionnelle, d'une séquence d'observations d'une grandeur mesurée en fonction d'une autre. La mesure de la température en fonction du temps par exemple, la figure 2.1 présente cet exemple de donnée fonctionnelle de température ($^{\circ}\text{C}$) mesurée en fonction du temps (mensuellement) dans le golf du

Mexique.

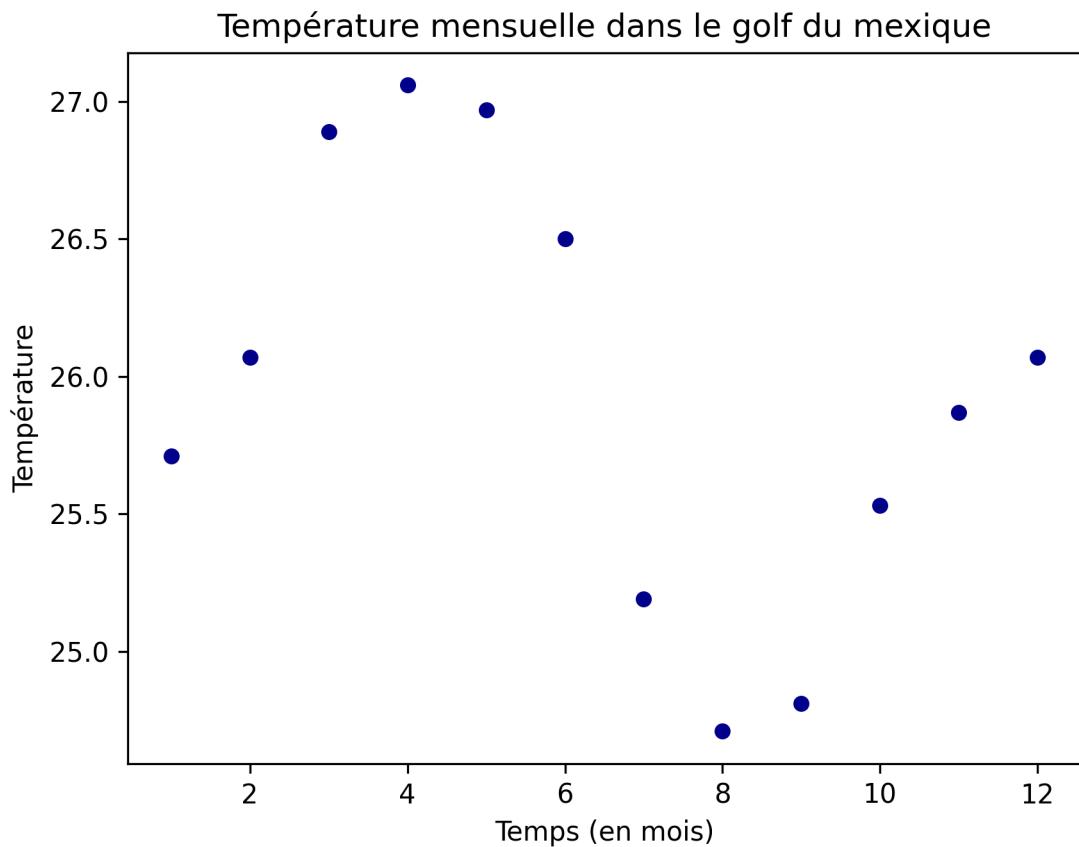


FIGURE 2.1 – Température mensuellement mesurée dans le golf du Mexique. Il s’agit d’une unique observation fonctionnelle (température en fonction du temps). Cette forme de donnée constitue la brique de l’analyse de donnée fonctionnelle.

Une telle donnée est appelée dans le domaine de l’analyse de données fonctionnelles une observation, et on entend compris dans cette observations les 12 points de temps auxquels elle est mesurée. On note alors cette observation

$$x = (x_t)_{1 \leq t \leq T} = (x_t)_{t \in [0, T]} = (x_t)_{t \in [T]}$$

où $T - 1$ est le nombre total de temps auxquels est mesurée X et $[T]$ est l’ensemble $1, 2, \dots, T$. On utilisera pour des raisons de simplification d’écriture cette notation par la suite. On peut également noter ce vecteur $[0 : T]$, notation vectorielle cohérente avec les notations machine, si l’est nécessaire de préciser l’indice de

départ (supposé 0 autrement). On peut choisir arbitrairement d'indexer les t de 0 à T , en sachant que si les mesures de la donnée fonctionnelle X sont faites dans un intervalle $[a : b] \subseteq \mathbb{R}$ quelconque à des temps $a = t_0, t_1, t_2, \dots, t_K = b$, le changement de variable par translation

$$\forall k \in [K], j = t_k - a, T = b - a$$

nous permet d'obtenir une indexation j de 0 à T .

Si l'on se donne un jeu de données de n observations fonctionnelles toutes mesurées aux mêmes temps $t_k \in [T], \forall k \in [K]$, ces observations peuvent s'écrire sous forme d'une matrice :

$$\mathcal{X} = \begin{bmatrix} x_1(0) & x_1(t_1) & x_1(t_2) & \cdots & x_1(T) \\ x_2(0) & x_2(t_1) & x_2(t_2) & \cdots & x_2(T) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_p(0) & x_p(t_1) & x_p(t_2) & \cdots & x_p(T) \end{bmatrix}$$

Il est cependant à noter que les (t_k) ne sont pas forcément régulièrement espacés, et que toutes les observations x_1, \dots, x_p ne sont pas forcément toutes mesurées aux mêmes temps. Dans un cadre général on note alors une observation $\forall i \in [1 : p], (x_i(t_{k_i}))_{k_i \in [K]}$ avec la suite t_k qui dépend de i . Pour chaque observation, l'ensemble des t_k sur lesquels elle est observée s'appelle une grille d'observation, et elle dépend de la donnée fonctionnelle dans un cas général. Il se peut que dans un même jeu de donnée les grilles soient les même pour tous les x_i , cependant en pratique cela n'a aucune raison particulière d'arriver et considérer le cadre général fait partie des enjeux majeurs de l'analyse de données fonctionnelles.

L'outil [scikit-fda](#) [5], bibliothèque python d'analyse de données fonctionnelles, permet la visualisation de ce type de jeu de données peu importe l'espacement des mesures et peu importe les différences de mesures. La figure 2.2 représente exactement ce à quoi peut ressembler un tel jeu de données avec pour exemple classique le jeu de données Canadian Weather.

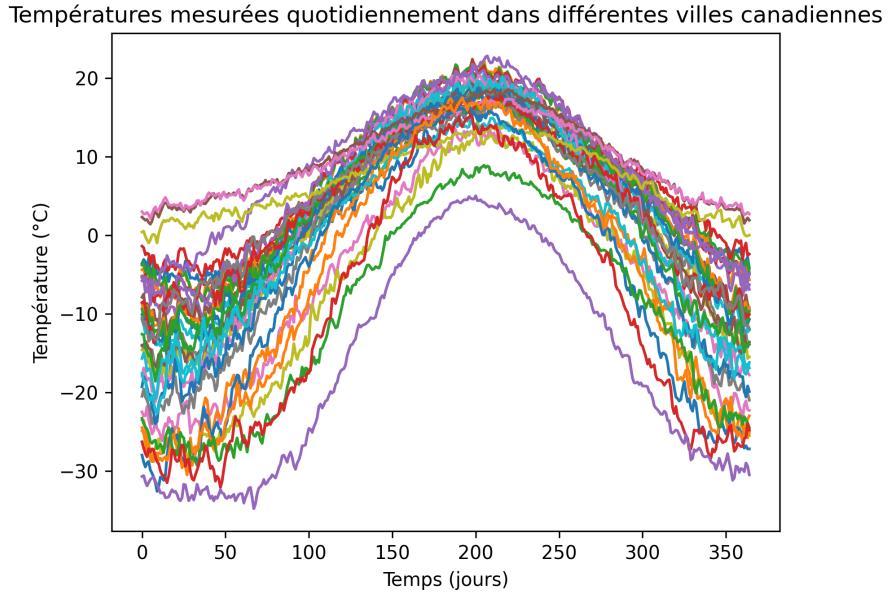


FIGURE 2.2 – Températures journalières dans 35 villes canadiennes. Ici les 35 observations sont des fonctions, propre à chaque ville. Les mesures ne sont pas toujours faites le même jour dans toutes les villes. La grille de mesures peut varier d'une ville à l'autre cependant la bibliothèque [scikit-fda](#) permet leur affichage sur un même graphe et traite chacune des suites de mesures (observations) indépendamment et chacune a sa propre grille t_0, t_1, \dots, T .

1.2 Représentation

Comme nous l'avons vu précédemment l'une des hypothèses majeurs de l'analyse de données fonctionnelles est la continuité et le caractère lisse des variables aléatoires fonctionnelles dont nous observons des réalisations. Pour travailler avec cette hypothèse, on s'occupe souvent de pré-traiter les données en considérant une représentation lisse (ou au moins $\mathcal{C}^2(\mathbb{R})$) de ces observations. Il existe deux grands types de représentation lisse : L'interpolation et le lissage. Dans le premier cas, l'hypothèse est faite que les données ne sont pas bruitées et que par conséquent les valeurs mesurées doivent être considérées comme appartenant au processus stochastique les ayant générées.

Les mesures échantillonnes des observations peuvent être interpolées de plusieurs manières, mais les méthodes les plus courantes sont les interpolations polynomiales ainsi que les interpolation par courbes de Bézier ou B-splines curves qui généralisent des polynômes et les ont surpassé en terme de maniabilité et de ressources informatiques [6] (chapitre 3). Dans le cas du lissage, on suppose que le processus de mesure comporte du bruit et que par conséquent les valeurs

observées n'appartiennent pas nécessairement au processus stochastique les ayant générées directement. Parmi les lissages les plus courants, la base des B-splines est également une des plus utilisées pour des données non-périodiques. Si les données présentent une périodicité, les lissages par une base de Fourier peuvent s'avérer plus intéressant que les lissages par B-spline, mais en pratique les bases de B-spline sont tellement flexibles qu'elles permettent tout de même une bonne représentation des fonctions périodiques. Dans les deux cas il s'agit de réduire la dimension des données qui, selon les hypothèses de la FDA, sont de dimension infinie. On peut exprimer ces fonctions dans une base de dimension infinie appelée base de Schauder telle que pour toute fonction f dont nos données sont un échantillonnage discret, on peut écrire

$$x(t) = \sum_{k=0}^{+\infty} c_k \Phi_k(t)$$

On peut tronquer cette expression à un K donné pour approcher les fonctions x et réduire leurs dimensions, cette méthode porte le nom d'expansion de base.

On s'intéresse dans notre projet à des données qui sont non-périodiques, en supposant que l'on peut retirer la périodicité en tant que pré-traitement de données temporelles, ce qui nous pousse à utiliser essentiellement le système de base de fonctions B-splines, qui sont définies par L polynômes joints de manière lisse en $\tau = [0 = \tau_0, \tau_1, \dots, \tau_L = T_0]$ une série de noeuds du vecteur $[T_0]$. L'ordre de ces polynômes, c'est-à-dire le nombre le degré le plus haut est également un paramètre des fonctions de cette base. Le nombre de fonctions de cette base est définie par l'ordre m des polynômes plus le nombre de noeuds intérieurs au domaine de définition (ou le nombre de noeuds moins 2). Par exemple, si l'on souhaite lisser notre observation de température dans le golf du Mexique, on peut choisir une base avec 6 noeuds et d'ordre $m = 4$ comme en figure 2.3.

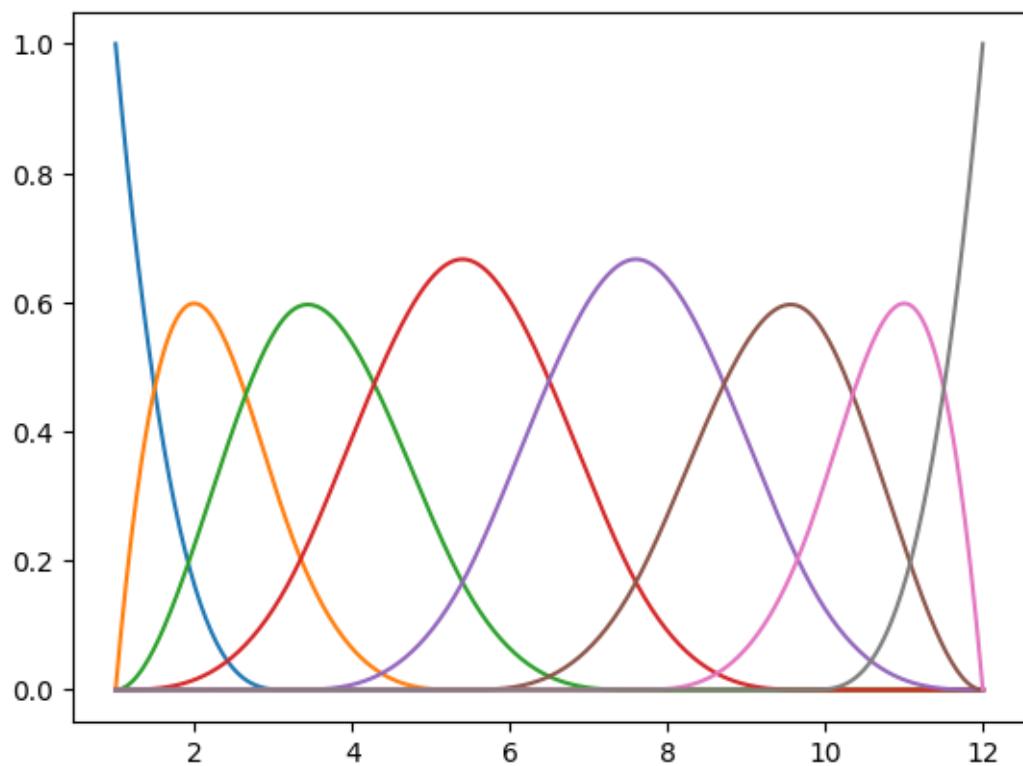


FIGURE 2.3 – Système de base de B-splines d’ordre 4 couvrant l’intervalle $[1, 12]$ avec 6 noeuds placés régulièrement sur la grille. Le nombre de fonctions de la base est alors de $4 + 6 - 2 = 8$, chaque couleur représente une fonction de la base choisie. On peut donc exprimer les données météorologiques du golf du Mexique vues précédemment comme des combinaisons linéaires de ces fonctions, ce qui aura pour effet de lisser les données. voir [2.4](#)

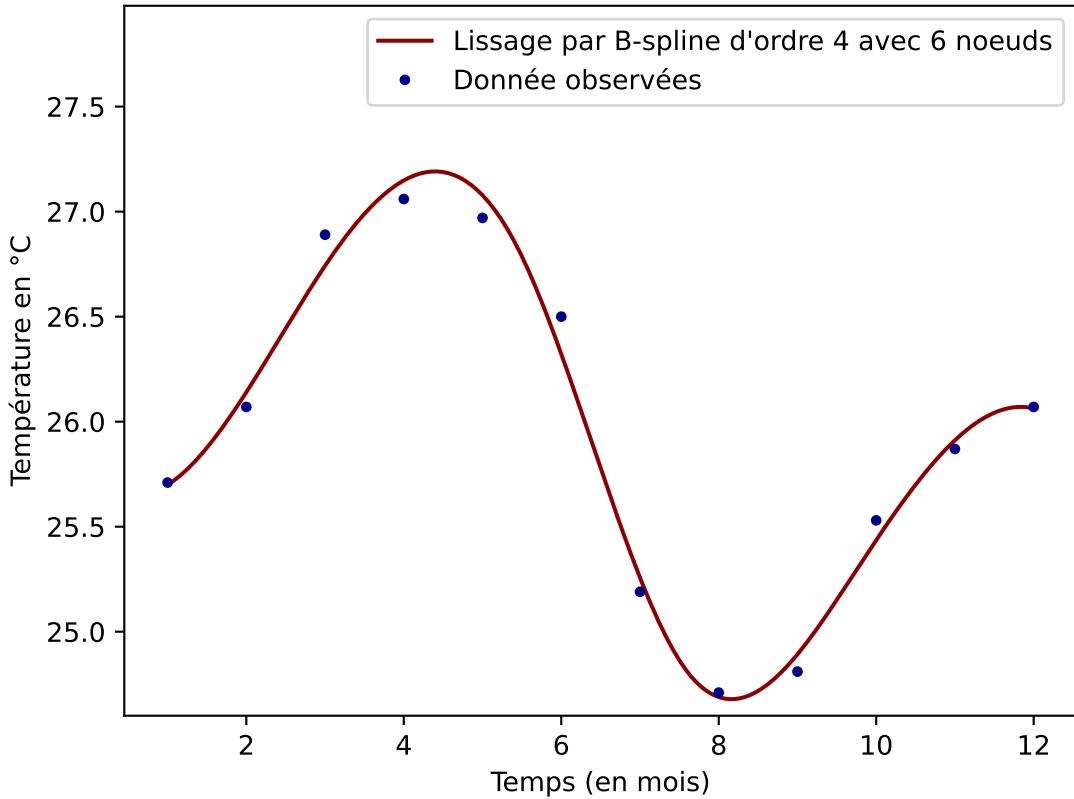


FIGURE 2.4 – Température dans le golf du mexique lissé par le système de base de B-splines d'ordre 4 couvrant l'intervalle $[1, 12]$ avec 6 noeuds vu en figure 2.3. Le nombre de fonctions de la base est alors de $4 + 6 - 2 = 8$, chaque couleur représente une fonction. On peut donc exprimer les données météorologiques du golf du Mexique vues précédemment comme des combinaisons linéaires de ces fonctions, ce qui aura pour effet de lisser les données.

1.3 Dérivées :

Parfois, les informations apportées par les variations d'un phénomène peuvent être capital dans la classification de données fonctionnelles. Prenons l'exemple de la mesure du poids d'un individu, et admettons que l'on essaie de prédire son état de santé, sain, ou malade, en fonction de la mesure de celle-ci. Il est imaginable que de savoir que la personne pèse un certain poids apporte beaucoup moins d'information sur son état de santé que les variations dans la mesure de son poids (par exemple une perte de masse corporelle importante en un temps réduit peut-être un signal d'alerte pour un médecin). C'est pourquoi il est important d'estimer les dérivées des variables fonctionnelles aléatoires à l'origine des processus

stochastiques observés. On peut raisonnablement faire l'hypothèse que la distribution des courbes de poids d'un malade possède des dérivées premières et secondes plus importantes que les fonctions de poids d'une personne saine. Un des avantages des représentations par B-splines vus dans la partie précédente est le fait que les observations une fois lissées admettent toutes des dérivées (au moins d'ordre m). L'apport informationnel de ces dérivées dépend de la tâche et des données, cependant il est à noter que les dérivées sont un outil classique de la FDA [6].

Puisque cet outil s'offre au statisticien FDA il est alors naturel de le considérer dans notre problématique, et nous allons voir que les solutions que nous apportons comprennent également l'intégration des dérivées des variables fonctionnelles aux modèles de réseaux de neurones, ce qui n'a pas encore été fait dans la littérature.

2 L'apprentissage automatique

2.1 Qu'est-ce qu'un modèle d'apprentissage

Dans cette partie, nous nous allons introduire et utiliser le formalisme développé dans (Shai Ben-David et Shai Shalev-Schwartz) [9] pour l'apprentissage automatique. Nous allons introduire la décomposition biais-complexité qui à l'origine des idées de recherche que j'ai mené à bout au cours de ce stage, car celle-ci est utile afin de mieux comprendre les avantages de notre modèle.

Supposons que l'on ait à notre disposition des fonctions issues du domaine médical, par exemple des données issues d'électroencéphalogrammes de différents patients atteints ou non de la maladie d'Alzheimer. On va alors, à partir de ces données fonctionnelles, essayer de prédire si un patient est atteint de la maladie ou non. Il s'agit d'une tâche dite de classification binaire. Les minimes variations dans les signaux électriques mesurés dans un cerveau sont pour l'œil humain très difficile à déceler, et encore plus difficile à interpréter. Cependant, on se dit qu'avec un modèle d'apprentissage à qui l'on donne de nombreuses données fonctionnelles étiquetées, il serait capable de déterminer les interactions non-linéaires entre les différentes "caractéristiques" des fonctions pour en déduire l'état de santé du patient. Le rôle d'un modèle d'apprentissage supervisé est d'estimer une fonction compliquée comportant des interactions non-linéaire entre les valeurs observées et l'entraînement avec de nombreuses données étiquetées permet au modèle de "converger" vers une fonction de prédiction optimale. En réalité cette manière informelle d'aborder le problème peut s'exprimer formellement. Commençons par définir le cadre formel de l'apprentissage statistique en se munissant de plusieurs notions :

- **L'ensemble domaine** : Il s'agit de l'ensemble \mathcal{X} que l'on souhaite étiqueter, on peut considérer \mathcal{X} comme les variables dépendantes, ou les prédicteurs. Il s'agit souvent en pratique d'un ensemble de caractéristiques, quantitatives ou catégorielles. Dans le cadre de notre problématique , il s'agira de

données fonctionnelles. On suppose que les objets de \mathcal{X} sont munis d'une loi de probabilité \mathcal{D} que l'on essaiera d'apprendre avec le modèle.

- **L'ensemble réponse :** Il s'agit de l'ensemble des \mathcal{Y} , aussi appelés labels (étiquettes) ou encore variable dépendante, c'est la variable que l'on essaie de prédire à partir de \mathcal{X} . Dans le cadre de la classification binaire par exemple, pour la classification des patients en sain ou malade on aurait $\mathcal{Y} = \{-1, 1\}$ dans le cadre de la régression on aurait $\mathcal{Y} = \mathbb{R}$. Dans la mesure du possible, on aimerait que le modèle d'apprentissage puisse labelliser correctement autant d'objets de l'ensemble domaine que possible. Autrement dit, si il existe une fonction $f : \mathcal{X} \mapsto \mathcal{Y}$ qui est la fonction "correcte" de labellisation inconnue du modèle d'apprentissage, on aimerait que le modèle approche au mieux cette fonction, si on lui donne suffisamment de données pour s'entraîner.
- **Les données d'entraînement :** Il s'agit d'une suite finie S de m paires de la forme $S = (X_i, y_i)_{i \in [|1, m|]}$. Malgré le nom trompeur d'ensemble, il s'agit bien d'une suite, le mot ensemble est un abus de langage issu du fait que les objets de cette suite sont tirés parmi deux ensembles : le domaine et les réponses.
- **La sortie de l'apprentissage :** Le but du modèle après l'apprentissage est de nous donner une fonction $h : \mathcal{X} \mapsto \mathcal{Y}$ on appelle h un classifieur, un prédicteur ou une hypothèse. Lorsque h est issue d'un algorithme d'apprentissage A entraîné sur un ensemble d'entraînement S , on note alors $h_{AS} = A(S)$.
- **Mesure d'erreur :** La mesure d'erreur d'un modèle est définie comme la probabilité que le modèle se trompe en classifiant des objets de l'ensemble \mathcal{X} . Il s'agit de la probabilité de tirer un $x \in \mathcal{X}$ au hasard et d'avoir $h(x) \neq f(x)$. Formellement, la perte L du modèle est définie comme suit :

$$L_{f,D}(h) = \mathbb{P}_{x \sim \mathcal{D}}(h(x) \neq f(x)) \quad (2.1)$$

Et étant donné un ensemble d'entraînement S la perte après apprentissage peut se réécrire de cette manière :

$$L_{S,f,D}(A(S)) = \mathbb{P}_{x \sim \mathcal{D}}(A(S)(x) \neq f(x)) \quad (2.2)$$

Et si l'on a un ensemble fini de données dans \mathcal{X} , alors on peut définir la perte empirique (ou risque empirique) comme une approximation de cette perte :

$$L_S(h) := \frac{\text{Card}\{i \in [|1, m|], A(S)(x_i) \neq y_i\}}{m} \quad (2.3)$$

La perte que nous utiliserons en pratique une perte qui peut s'appliquer même si y peut appartenir plusieurs catégories et pas seulement 2, il

s'agit de l'entropie croisée empirique définie comme telle :

$$L_S(h) := -\frac{1}{m} \sum_{i=1}^m [y_i \ln(A(S)(x_i))] \quad (2.4)$$

Cependant, si la tâche à effectuer est une régression et non une classification, approcher la perte de cette manière peut ne pas fonctionner, car numériquement, si $\mathcal{Y} = \mathbb{R}$, alors $\forall i, A(S)(x_i) \neq y_i$, et on préférera mesurer le risque par une mesure d'écart quadratique moyen de la prédiction à la réponse pour mesurer l'efficacité de l'hypothèse h :

$$L_S(h) := \frac{\sum_{i=1}^m (h(x_i) - y_i)^2}{m} \quad (2.5)$$

Le modèle d'apprentissage est donc l'ensemble de toutes ces notions, on entend en réalité par modèle d'apprentissage (ou learner), à la fois l'algorithme A qui va nous permettre d'approcher la fonction f optimale, et la fonction $h = A(S)$ le prédicteur qui résulte du dit algorithme. Le choix de l'ensemble d'entraînement, son découpage en lots (batchs) ainsi que la mesure de la perte que l'on choisit sont ce que l'on appelle des hyper-paramètres du modèle, et ces hyper-paramètres sont eux même des paramètres de l'algorithme A qui comprend le choix de la famille de prédicteur ainsi que la phase d'apprentissage et la mesure de succès.

2.2 Décomposition biais-complexité

Pour que l'apprentissage soit effectif, la fonction de risque empirique définie précédemment est ce que l'on cherche à minimiser au cours de l'entraînement. Un tel procédé s'appelle la règle du MRE (ERM en anglais) pour minimisation du risque empirique. Cependant minimiser le risque empirique peut conduire à du sur-apprentissage [9] (chapitre 3). Pour régler ce problème, on préfère limiter la taille de l'ensemble de fonctions h que le modèle peut apprendre. Cet ensemble s'appelle une classe d'hypothèses noté \mathcal{H} . Formellement, étant données une classe \mathcal{H} et un ensemble d'entraînement S , on définit le MRE comme suit :

$$MRE_{\mathcal{H}}(S) \in \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$$

Lorsque l'on réduit la classe d'hypothèses de cette manière, alors on crée un biais dans hypothèses h que notre modèle pourra apprendre. On peut en réalité décomposer l'erreur d'apprentissage du modèle en biais et complexité. Cette décomposition est fondamentale pour la suite car elle est celle qui a guidé notre recherche dans la création et le choix des modèles pour répondre à notre problématique d'analyse de données fonctionnelles. Soient S un ensemble d'entraînement et soit h_S un $RME_{\mathcal{H}}$. Alors on peut réécrire la perte empirique de la manière suivante :

$$L_D(h_S) = \varepsilon_{biais} + \varepsilon_{est} \text{ avec } \varepsilon_{biais} := \min_{h \in \mathcal{H}} L_D(h) \text{ et } \varepsilon_{est} = L_D(h_S) - \varepsilon_{biais} \quad (2.6)$$

L'erreur ε_{est} correspond à l'erreur d'estimation, c'est à dire qu'elle est complètement issue du fait que le risque empirique n'est qu'une approximation du risque théorique. L'erreur ε_{biais} quand à elle est issue du fait que potentiellement la fonction f que l'on cherche à estimer n'appartienne pas à la classe \mathcal{H} d'hypothèse du modèle que nous utilisons pour l'apprendre. Ce compromis biais-complexité est à l'origine des modèles que nous avons développé au cours de ce projet. En effet l'utilisation d'un modèle plus "simple", avec une classe d'hypothèse \mathcal{H} moins grande peut s'avérer meilleur en pratique qu'un modèle plus complexe. En réduisant la classe d'hypothèse, on fait le pari que le gain gagné en biais écrase l'erreur que nous d'approximation de f que nous ajoutons à la perte, on espère également qu'il sera plus "facile" de trouver une hypothèse convenable qu'en cherchant parmi de trop nombreux classificateurs possibles sans jamais converger vers un optimal. Un bon choix de classe \mathcal{H} peut se traduire en une convergence plus rapide ou encore en un besoin moins important en données d'entraînement, ce qui peut s'avérer crucial en pratique.

3 Le réseau de neurones artificiel

3.1 Définition d'un réseau de neurones artificiel

Parmi les classes de modèles d'apprentissage définis dans la section ??, le réseau de neurone est un modèle particulièrement large en terme de classe d'hypothèse \mathcal{H} qu'il peut apprendre. L'idée est de sur-paramétrier une tâche bien précise en composant des fonctions linéaires paramétriques avec des fonctions non-linéaires non paramétriques, et d'ajuster les paramètres linéaires (aussi appelés poids) en remontant le gradient de la perte de cette immense fonction composée avec les vraies étiquettes pour optimiser la sortie recherchée. Il s'agit en réalité d'une des classes de modèles les plus expressives, c'est à dire que la classe d'hypothèse \mathcal{H} est parmi les plus grandes de tous les modèles d'apprentissage. Nous reviendrons plus tard sur la puissance d'expression de cette classe de modèles. Commençons d'abord par définir le réseau de neurone formellement. Plus particulièrement, on définit ici la structure du réseau de neurones à propagation avant (feedforward).

Il est défini comme un graphe acyclique $\mathcal{G} = (V, E)$ avec V l'ensemble des couches du graphe et E les arrêtes du graphe. E est l'ensemble des coefficients du modèle, il est muni d'une fonction de poids $w : E \mapsto \mathbb{R}$, et l'on appelle un élément de E un neurone. V est une union disjointe de couches $V = \bigsqcup_{c=1}^C V_c$, où C est le nombre de couches du graphes. Si $n \in \mathbb{N}$ est la taille de l'espace d'entrée, alors V_0 est comporte $n + 1$ neurones. Un cas spécifique est le réseau de neurone fully-connected ou MLP, il s'agit d'un cas particulier de graphes dans lesquels tous les neurones de chaque couche sont connectés aux neurones des couches suivantes et précédentes. La figure 3.1 est un exemple simple à deux couches de réseau complètement connecté. Les fonctions non-linéaires sont appelées

activation notées $\sigma : \mathbb{R} \mapsto \mathbb{R}$, les plus courantes sont les fonctions signe, $\sigma(x) \mapsto \text{signe}(x)$, $\text{ReLU}_r(x) \mapsto \max(x, 0)$, sigmoïde, $\sigma(x) \mapsto 1 / (1 + \exp(-x))$. On procède alors au calcul couche par couche pour obtenir la sortie du réseau de neurones après y avoir entré le vecteur X . Pour un c donné, et un élément $v_c, j \in V_c$ de la $c - ième$ couche, on peut écrire l'entrée e_{c-1} de v_c ainsi :

$$e_{c-1,j} = \sum_{r, (v_{c-1}, v_c, j) \in E} w(v_{c-1}, v_c, j) o_{c-1,r}(x)$$

où $o_{c,j} = \sigma(e_{c,j}(x))$

On peut écrire la fonction du réseau de neurones NN_w comme une fonction composée de toutes les sorties de toutes les couches.

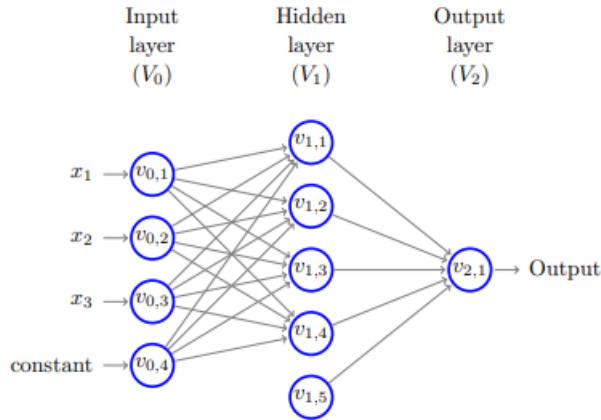


FIGURE 2.5 – Réseau de neurones à deux couches (dont une couche cachée). La couche d'entrée V_0 est de dimension 4, à savoir les trois dimensions de l'entrée X ainsi que la constante (appelée biais). Il s'agit d'un exemple de réseau de neurones complètement connecté (fully-connected ou FCNN) car chaque neurone est connecté à tous les neurones de la couche suivante et précédente. (Source : *Understanding Machine Learning* : Shai Shalev-Shwartz and Shai Ben-David 2014)

Bien évidemment un réseau de neurones ne serait rien sans sa capacité d'apprentissage. C'est pourquoi un tel graphe est également muni d'un optimiseur, c'est à dire d'un algorithme d'optimisation qui permet d'adapter chacun des poids de E de manière à minimiser la perte empirique. Le réseau de neurone étant une fonction composée de fonctions différentielles, il est alors lui même différentiable, et en pratique on utilise un algorithme tel que la descente de gradient stochastique ou l'estimation de moment adaptative (Adam), qui effectue la propagation arrière de la perte. On entend propagation arrière ou rétro-propagation (backpropagation)

dans la littérature) la méthode qui consiste à chaque couche du réseau de neurone de calculer le gradient en les poids de E , d'effectuer la descente de gradient de la couche précédente, et ainsi de suite jusqu'à adapter les poids de la première couche. Cette méthode est munie d'un pas d'apprentissage (noté ρ et souvent faible, de l'ordre de 10^{-4}) qui détermine la force de l'adaptation des poids à chaque étape de l'algorithme. L'algorithme que nous utilisons le plus souvent en pratique pour l'optimisation des différents modèles est la descente de gradient stochastique dont voici l'algorithme :

Algorithm 1 Descente de gradient stochastique, *Stochastic Gradient Descent* (SGD)

Require: Pas d'apprentissage ρ , Nombre d'époques N , paramètres du modèles θ , NN réseau de neurones définie comme fonction composée de fonctions admettant un gradient, m le nombre de mini lots d'entraînement, fonction de perte empirique L comme définie en équation 2.2

- 1: Paramètres initiaux θ_0
 - 2: **for** $k \leftarrow 1$ to N **do**
 - 3: **for** $i \leftarrow 1$ to m **do**
 - 4: Tirage d'un ensemble d'entraînement S : $(x^{(i)}, y^{(i)})$
 - 5: Calcul du gradient de la perte :
- $$\nabla_{\theta} \leftarrow \nabla_{\theta} L \left[NN_{\theta} \left(x^{(i)} \right) - y^{(i)} \right]$$
- 6: Actualiser les poids : $\theta \leftarrow \theta - \rho \nabla_{\theta}$
-

D'autres variantes de cet algorithme, avec des estimations du jacobien de la fonction NN_{θ} ou avec un pas d'apprentissage adaptatif seront utilisés en pratique lors de nos expérimentations, chaque modèle nécessitant un algorithme d'apprentissage particulier sur chaque jeu de données.

3.2 Un estimateur de fonction universel

Intéressons nous désormais à la puissance expressive des réseaux de neurones artificiels, c'est à dire à la classe de fonctions qu'ils sont capables d'apprendre. Cette classe de modèle d'apprentissage est en réalité un estimateur de fonction binaire universel, nous nous basons pour avancer ce fait sur la démonstration de Shai Ben David et Shai Shalev Shwartz, qui montre que toute fonction d'entrée discrète binaire de classification en deux classe peut être estimée par un graphe à 2 couches comme défini précédemment.

Théorème d'approximation universelle : Pour tout $n \in \mathbb{N}$ il existe un graphe (V, E) de profondeur 2 (à une couche cachée) tel que :

$$\forall f : \{\pm 1\}^n \mapsto \{\pm 1\}, f \in \mathcal{H}_{V,E,sign}$$

Démonstration :

Soit $n \in \mathbb{N}$, et soit $f : \{\pm 1\}^n \mapsto \{\pm 1\}$. On construit un graphe avec $|V_0| = n + 1$, $|V_1| = 2^n + 1$, $|V_2| = 1$. On commence par se donner tous les u_1, u_2, \dots, u_K vecteurs de $\{\pm 1\}^n$ tel que $\forall i \in [1 : K]$, $f(u_i) = 1$. Puis on observe que pour tout $i \in [k]$ le produit scalaire $\langle x, u_i \rangle = n \Leftrightarrow x = u_i$ et que $x \neq u_i \Rightarrow \langle x, u_i \rangle \leq n - 2$. Alors, la fonction suivante $g_i(x) := signe(\langle x, u_i \rangle - n + 1)$ est une indicatrice de l'égalité entre u_i et x . En effet si $x = u_i$ alors $\langle x, u_i \rangle = n \Rightarrow g_i(x) = 1$. Si $x \neq u_i$, $\langle x, u_i \rangle \leq n - 2 \Rightarrow n + \langle x, u_i \rangle \leq 0$. Il suffit alors d'ajuster les poids entre V_0 et V_1 pour que chaque neurone de la couche V_1 applique la fonction g_i . Formellement, on peut choisir la fonction de poids de E tel que $\forall i \in [1 : K]$, $v_i \in V_0$, $v_{1,i}(x) = g_i(x)$. En remarquant que f est la fonction composée de toutes les sorties de toutes les couches, donc g appliquée à la somme des $(g_i + k) - 1$ et peut s'écrire ainsi :

$$f(x) = signe \left(\sum_{i=1}^k g_i(x) + k - 1 \right)$$

f peut finalement s'exprimer complètement à partir du graphe.

Remarques :

Ce résultat est extrêmement puissant, en raison du fait que toutes les fonctions informatiques sont codées de manière binaire. Cela signifie que le réseau de neurones artificiel a potentiellement le pouvoir d'approcher n'importe quelle fonction en machine. Il est cependant à noter que ce résultat théorique ne rend pas compte de la difficulté pratique de l'approximation de fonctions lorsque n devient grand, la taille du graphe utilisé dans la démonstration augmentant de manière exponentielle en n . Nous sommes désormais munis d'une classe d'apprentissage très puissante mais dont la complexité peut augmenter de manière exponentielle en fonction de la taille de l'ensemble que l'on cherche à analyser avec. Revenons-en à l'analyse du compromis biais complexité, on peut dire que le réseau de neurone artificiel a une très grande classe d'hypothèse \mathcal{H} et que son erreur d'approximation est très faible. Cependant cette complexité très grande peut mettre en péril l'apprentissage en pratique étant donné le nombre possible d'hypothèses à parcourir afin d'en atteindre un qui serait optimale. C'est pourquoi il est parfois préférable de réduire la complexité des modèles pour atteindre un optimal plus régulièrement et plus efficacement. En réduisant par exemple le nombre de connexions entre les neurones, il se peut que la perte en terme d'erreur d'approximation en augmentant le biais soit effacée par le gain en complexité que l'on obtient.

3 L'opération de convolution

3.1 Convolution et probabilités

Supposons que l'on ait X et Y deux variables aléatoires continues indépendantes, de fonctions de densité f et g respectivement. Soit $\varphi : \mathbb{R} \mapsto \mathbb{R}^+$ une application mesurable. Soit $S := X - Y$. Alors on a :

$$\begin{aligned} \mathbb{E}[\varphi(S)] &= \mathbb{E}[\varphi(X - Y)] \underset{\substack{=} \\ \text{Théorème de transfert}}{=} \int_{\mathbb{R}} \int_{\mathbb{R}} \varphi(x-y) d\mathbb{P}_{X,Y} \underset{\substack{=} \\ X \perp Y}{=} \int_{\mathbb{R}} \int_{\mathbb{R}} \varphi(x-y) d\mathbb{P}_X d\mathbb{P}_Y \\ &= \int_{\mathbb{R}} \int_{\mathbb{R}} \varphi(x-y) f(x) g(y) dx dy \underset{\substack{=} \\ \begin{cases} y = x + s \\ ds = dy \end{cases}}{=} \int_{\mathbb{R}} \int_{\mathbb{R}} \varphi(s) f(x) g(x+s) dx ds \\ &= \int_{\mathbb{R}} \varphi(s) \underbrace{\int_{\mathbb{R}} f(x) g(x+s) dx}_{=\Phi(s)} ds = \int_{\mathbb{R}} \varphi(s) d\mathbb{P}_S \quad \text{où } d\mathbb{P}_S = \Phi(s) d\lambda(s) \end{aligned}$$

La fonction de densité de la loi de S n'est autre que la fonction suivante

$$\boxed{\Phi(s) = \int_{\mathbb{R}} f(x) g(x+s) dx} \quad (2.7)$$

Il s'agit de la fonction de corrélation croisée entre les densités f et g . C'est cet opérateur qui est utilisé lorsque l'on parle de convolution, (dans le cas discret) entre deux vecteurs dans les réseaux de neurones artificiels. Dans les librairies d'apprentissage profond c'est aussi cette opération qui est appelée convolution, et c'est la corrélation croisée de l'équation 2.7 que dans la suite de ce travail appelleront par abus de langage, mais surtout pour rester cohérent avec la littérature, l'opération de convolution [3]. La résultante de l'opération de convolution entre f et g est appelée la convolée, et est notée $(f * g)$. Dans le cas numérique discret g est appelée un filtre ou un noyau, et la fonction f est la donnée d'entrée de la convolution. Dans le cas discret, f et g ne sont plus des densités mais des tenseurs ou des matrices. Le but d'un tel modèle d'apprentissage est d'apprendre g pour extraire de f ses traits qui permettent de la classifier. Dans le cadre de la classification d'image par exemple, f est une fonction de deux dimensions qui à chaque pixel (ou position *hauteur* \times *largeur*) associe un chiffre définissant l'intensité de chaque canal de couleur primaire (rouge, bleu et vert). Dans le domaine de la classification d'image, les réseaux de neurones convolutifs ont fait leurs preuves comme étant très puissant et précis, exemple pour la classification de personnages (S Ben Idriss *et al*) [1]. Le principe est d'apprendre une multitudes

de noyaux g pour capter des propriétés ou particularités locales des images qui permettent de les discriminer en classes. En effet, si l'on considère que l'image f est une approximation d'un processus aléatoire et qu'il existe une variable aléatoire X (par exemple la classe de l'image) admettant f comme densité, alors il convient d'apprendre les meilleurs Y tels que $X - Y$ est discriminant, c'est à dire qu'il permet de précisément classifier les X . Ce procédé est très analogue au processus de filtrage en théorie du signal, mais dans lequel on aurait incorporé l'optimisation des filtres pour obtenir un résultat objectif (par exemple minimiser une perte théorique comme définie en 2.1). Cette approche est fondamentale dans l'intuition du professeur C.Beaulac, qui souhaite prendre en compte le fait que les données fonctionnelles présentent de fortes corrélations par voisinage. Ainsi apprendre les propriétés locales discriminantes de fonction s'inscrit parfaitement dans la problématique de la FDA.

3.2 La couche de convolution

Nous avons vu alors l'intuition à l'origine de l'introduction de cette opération dans les modèles d'apprentissages statistiques, mais comment ce principe intervient-il-exactement ? Dans le cas de l'apprentissage statistique, et surtout dans le cadre de l'analyse d'image où cette architecture est énormément utilisée, on parle de réseau de neurone convolutif lorsqu'une ou plusieurs couches du réseau de neurones effectuent l'opération de corrélation croisée entre un noyau \mathcal{K} , aussi appelé kernel ou filtre, et une donnée d'entrée, comme une image codée en RGB pixel par pixel par exemple, avant l'utilisation d'un bloc fully-connected. Le kernel est constitué des poids qui seront appris lors de la propagation arrière de la perte du réseau de neurones. Le nombre de couleurs de l'image (1 si noir et blanc et 3 si RGB) s'appelle des canaux (ou channels en anglais). Comme vu précédemment, la sortie de l'opération entre deux fonctions dans le cas continu donne une fonction également. Nous allons voir que dans le cas discret, c'est également le cas, c'est à dire que cette opération a pour effet de conserver la nature de la variable d'entrée. La sortie de la convolution d'une image sera donc une image, celle d'une fonction une fonction également, et c'est en partie là d'où vient l'intuition du professeur C. Beaulac d'utiliser la convolution dans l'analyse de données fonctionnelles. En effet, conserver la structure de la donnée d'entrée de réseau de neurones plus longtemps semble cohérent pour classifier des données fonctionnelles par exemple.

Formalisons cette opération dans le cas discret. Supposons que l'on a un objet vectoriel d'entrée $X = (x_1, \dots, x_{T_0})$ et que l'on se donne un noyau $(\mathcal{K})_K = \mathcal{K}(1), \dots, \mathcal{K}(K)$ de taille K , l'opération de corrélation croisée entre \mathcal{K} et X s'écrit ainsi :

$$\forall t \in [T_1], (X * \mathcal{K})(t) = \sum_{k=1}^K \mathcal{K}(k) X(t+k) \quad (2.8)$$

Cette opération a pour effet de faire "glisser" le noyau \mathcal{K} sur l'entrée X et d'effectuer le produit scalaire entre toutes les valeurs de \mathcal{K} et les valeurs de X face auxquels elles sont superposées. L'opération comme définie ci-dessus est en fait un cas particulier de celles que peuvent effectuer les réseaux de neurones convolutifs. La couche de convolution comporte en effet plusieurs paramètres ajustables qui permettent de régler le filtre, ses propriétés, son déplacement sur le vecteur d'entrée, ainsi qu'un biais b qui sera appris. La taille du filtre est notée K , le pas avec lequel se déplace le noyau sur les données s'appelle l'enjambée ou le *stride* et est noté s . Il est également possible de définir l'écart entre les points de X qui sont pris en compte pour le produit scalaire avec \mathcal{K} , ce paramètre est appelé dilatation et on le notera d , il sera très important par la suite dans le cadre de la généralisation de cette opération. Le dernier paramètre important de cette couche est le rembourrage ou *padding*, il s'agit de l'ajout de zéros de part et d'autre de la donnée d'entrée lors de l'opération, et il est noté p . Une visualisation de l'effet des paramètres sur l'opération de convolution dans le cadre d'une image en deux dimensions est disponible à ce [lien](#).

De manière évidente, toutes les entrée dans X qui sont en dehors du domaine de X sont posés comme étant 0 (par exemple si $T_0 = 12$ et $p = 0$ alors $X(-1) = X(13) = X(14) = \dots = 0$). On conserve alors uniquement dans le vecteur de sortie final de l'opération les termes qui ne sont pas nuls par définition, c'est-à-dire les termes pour lesquels le noyau multiplie au moins un élément de X , on ne s'autorise le noyau à sortir du vecteur qu'avec le rembourrage. De ce fait on en déduit alors la dimension de sortie T_1 de cette opération en fonction de ces paramètres :

$$T_1 = \#C(X) = \left\lfloor \frac{T_0 + 2 \times p - d \times (K - 1)}{s} + 1 \right\rfloor \quad (2.9)$$

Il s'agit en résumé du "nombre d'étapes tel que le noyau rentre en entier dans X si l'on le déplace de s à chaque étape". La figure 2.6 montre un exemple simple de cette opération appliquée à une observation fonctionnelle.

FIGURE 2.6 – Couche de convolution appliquée à une donnée fonctionnelle. On reprend notre exemple de donnée fonctionnelle de température dans le golf du Mexique. On lui applique un noyau de taille 5 avec une dilatation de 1, pas de rembourrage et une enjambée de 1. La sortie de cet objet est également une donnée fonctionnelle, mais de taille 8.

Sachant une couche C munie d'un noyau \mathcal{K} et de ses paramètres (K, s, p, d) on notera alors l'opération de l'application de la couche C à une entrée X comme étant l'opération définie ci dessous :

$$\forall t \in [T_1], C_{\mathcal{K}}(X)(t) := b + \sum_{k=0}^{K-1} \mathcal{K}(k) X(ts + (k \times d)) \quad (2.10)$$

Enfin, il faut noter qu'en pratique on entraîne une multitude de noyaux simultanément, et pas seulement un seul. Les résultats vus précédemment s'appliquent pour tout noyau d'une couche de convolution. Dans un réseau de neurones convolutif, on applique plusieurs couches de convolution successivement, souvent en augmentant le nombre de noyaux et en réduisant la dimension de l'objet d'entrée à chaque couche, avant d'envoyer la sortie finale de dernière couche de convolution dans un réseau de neurones dense souvent d'une seule couche pour prédire la sortie finale, par exemple des classes.

3.3 Propriétés de la convolution

Nous avons vu comment fonctionne une couche de convolution, et nous allons maintenant énoncer les propriétés d'une telle opération au sein d'un réseau de neurones en tant que modèle d'apprentissage. Nous nous baserons pour cela en

grande partie sur le chapitre 9 du livre *Deep Learning* (Ian Goodfellow *et al.*) sur la convolution.

- **Partage de paramètres :** Les poids du noyau sont appliqués à l'ensemble du vecteur X contrairement au cas du réseau de neurone dense. En effet puisque le noyau glisse sur l'entièreté de X alors les poids appliqués au début du vecteur sont les mêmes que ceux appliqués aux dernières composantes du vecteur. Cela a pour effet de relever les caractéristiques locales importantes et discriminantes des objets d'entrée lors de l'apprentissage des noyaux. L'apprentissage de noyaux s'appliquant à un voisinage a , d'un point de vue heuristique, l'effet de capturer des propriétés locales de l'objet d'entrée, car chaque filtre s'applique à un voisinage des points de l'objet d'entrée. On a donc l'espoir qu'apprendre ces filtres aurait donc pour effet d'apprendre les poids tels que les effets dans un voisinage de points de X sont repérés et aident à sa classification, étant donné l'hypothèse structurelle de corrélation forte des voisinages dans les données fonctionnelles. La figure 2.7 présente une visualisation de ce phénomène.
- **Connectivité creuse :** Le nombre total de paramètres à apprendre est drastiquement réduit par cette méthode. Si l'on regarde cet aspect de la convolution avec notre point de vue biais-complexité, alors le biais est augmenté car la classe d'hypothèse étant plus petite que celle du MLP, alors il est plus difficile en théorie d'atteindre l'estimateur optimal. On gagne en revanche du côté de l'erreur d'estimation car la complexité du modèle est plus faible, par conséquent il est plus rapide de parcourir \mathcal{H} en pratique à la recherche d'une hypothèse satisfaisante.
- **Préservation de la structure :** Dès l'entrée dans un réseau de neurones complètement connecté, l'objet d'entrée est "écrasé" (on parle d'opération de Flatten), c'est à dire qu'on le transforme en un vecteur une dimension et toute information structurelle est perdue. Par exemple, un ensemble de n images de $p \times p$ pixels est transformée immédiatement en un vecteur de taille $n \times p^2$, dans lequel l'ordre n'est pas pris en compte et que le réseau de neurone va devoir apprendre à classifier comme un vecteur immense. Dans la convolution, il s'agit d'affiner ainsi que de réduire la dimension de l'objet d'entrée avant de le passer au "mixeur vectoriel", ainsi une image . Dans la problématique d'analyse de donnée fonctionnelle, puisqu'il paraît que les données soient issues d'un processus lisse bruité, ce processus nous permettra de sur-ligner et conserver les "temps forts" (highlights) de ce processus au cours de plusieurs étapes de convolution et permettra d'en déduire la classe de la donnée fonctionnelle.

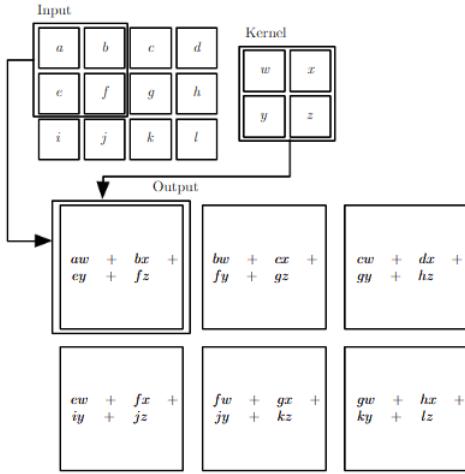


FIGURE 2.7 – Partage de paramètres : On voit que les mêmes paramètres sont utilisés au moins une fois pour chaque composante du vecteur d'entrée. Apprendre ces paramètres revient donc à apprendre les caractéristiques discriminantes locales de l'objet d'entrée. Source : *Deep Learning* (IanGoodfellow et al.) [3]

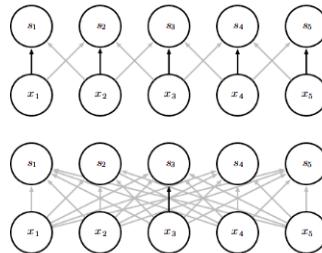


FIGURE 2.8 – Connectivité creuse : Chaque flèche correspond à un paramètre à apprendre. En premier on a l'exemple de la convolution avec une enjambée de 1, et en second, le cas de deux couches complètement connectées. On voit que le nombre de connexions dans le cas de la convolution est drastiquement réduit, la classe d'hypothèse \mathcal{H} est donc plus petite, ce qui permet de trouver plus facilement un optimal, mais augmente le biais. Source : *Deep Learning* (IanGoodfellow et al.) [3]

CHAPITRE 3

CONVOLUTION LISSÉE AJUSTABLE

1 Convolution sur données fonctionnelles :

Supposons tout d'abord que l'on soit munis d'un jeu de données fonctionnelles \mathcal{X} régulièrement espacées et mesurées. On peut alors appliquer la convolution sur les vecteurs qui sont tous de la même taille comme vu en figure 2.6. Les couches de regroupement par maximum (MaxPooling) ont également fait leurs preuves dans la littérature pour la reconnaissance d'images comme par exemple pour la reconnaissance de mouvement de mains [4]. Dans le cadre de classification, la convolution a pour but d'extraire des propriétés locales des données, ainsi l'extraction du maximum après coup semble intéressante pour classer ces dernières. C'est pourquoi après avoir appliqué une couche de convolution, on appliquera également une couche de MaxPooling 3.1.

FIGURE 3.1 – Couche de MaxPooling appliquée à la sortie de la convolution de la figure 2.6, cette couche a pour effet de mettre en lumière les variations dans les sorties de convolution, en pratique, cela a un impact positif sur les performances.

La combinaison d'une couche de convolution et d'une couche de Max Pooling appliquées au données brutes a déjà été mis en place par B.Thind *et al.* [14] sur les données "tecators", dans le cadre de la régression, données qui sont toujours régulièrement espacées. Dans notre travail, on a également commencé par considérer ce cas. En effet, le professeur C.Beaulac a eu l'idée d'utiliser la convolution sur les données fonctionnelles pour incorporer le fait que les données fonctionnelles sont fortement corrélées si elles sont proches (la taille d'un individu le jour j est très corrélée à sa taille le jour $j + 1$). Dans la même démarche, il était assez logique d'incorporer également les autres hypothèses de l'analyse de données fonctionnelles, surtout l'hypothèse de régularité des variables fonctionnelles sous-jacentes.

1.1 Problèmes de ce modèle :

La convolution étant définie comme une opération régulière, d'un filtre se déplaçant de manière constante (avec un enjambement fixé) sur les points de la courbe, les données observées irrégulièrement ne peuvent pas être analysés de la même manière par le même filtre se déplaçant de la même façon sur toutes les

observations.

En effet, admettons que les observations fonctionnelles soient mesurées de manière journalière mais pas exactement toutes le même jour pour des raisons pratiques, alors le noyau de la couche de convolution d'un réseau de neurone devrait "s'adapter" en taille et en enjambée selon les observations, ce qui rend impossible l'entraînement par lot.

Une solution naïve à ce problème est de créer une matrice de grande taille comportant en colonne tous les points de temps auxquels est mesurée au moins une des observations parmi le jeu de données, et de considérer les observations non mesurées en ces points comme étant 0. On obtient alors une matrice creuse de grande taille dont toutes les observations sont mesurées à des temps réguliers et sur laquelle il est possible d'appliquer la convolution telle qu'elle est implémentée dans [Pytorch](#). Cette approche semble en revanche déraisonnable car pouvant s'apparenter à l'une des pires formes d'imputation de données manquantes qui existe : l'imputation par zéros. Cela rajoute également de grandes variations artificielles au sein des fonctions que les noyaux de convolution pourraient interpréter et apprendre.

Quand bien même pourrait-on appliquer ces filtres en pratique car toutes les données comporteraient le même nombre de mesures, mais mesurées irrégulièrement, l'application de ces filtres sur des données irrégulièrement espacées reviendrait à apprendre des filtres qui ne "voient" pas les mêmes temps selon les données d'entrée. Cela supprimerait le principe de localité et la logique des filtres partageant les mêmes paramètres, ce qui va à l'encontre des idées qui nous poussent à utiliser la convolution.

De plus, la convolution en une dimension appliquée à des observations fonctionnelles brutes ne prend pas en compte l'hypothèse de base faite en analyse de données fonctionnelle qui est que les observations sont issus d'un processus lisse (ou du moins régulier) plus une erreur aléatoire (liée à l'erreur de mesure et à la nature aléatoire du processus). Habituellement en analyse de données fonctionnelles, on utilise comme vu dans la section ?? un lissage de la fonction pour capter au mieux le processus à l'origine des observations mesurées, en partant du principe que les observations en elles-mêmes comportent des erreurs. Appliquer la convolution directement sur les points observés implique de supposer que les données ne sont pas bruitées. Il semble plus réaliste de modéliser ces données comme étant lisses d'abord avant d'appliquer la convolution.

Nous avons désormais abordé toutes les notions nécessaires à l'introduction de notre modèle. Notre contribution est de répondre à cette problématique en utilisant des outils existants, et en outre passant les problèmes que nous venons d'évoquer. La couche d'entrée que nous avons développé a également l'avantage d'incorporer toutes les hypothèses de l'analyse de données fonctionnelles, ainsi que les dérivées des observations en plus de présenter des résultats prometteurs en terme de performance.

2 La convolution lissée ajustable : Description du principe

Le modèle que nous proposons est un bon compromis entre le réseau de neurones complètement connecté, dont la complexité est très grande et dont le biais est très faible, et la convolution unidimensionnelle dont le biais est plus grand mais la complexité faible. Nous appelons ce modèle proposé la convolution lissée ajustable, *Tunable Smoothed Convolutional Neural Network* (TSCNN).

L'idée est de combiner les outils d'analyse de données fonctionnelles, tel le lissage, avec la convolution unidimensionnelle : On commence en première couche par lisser chacune des observations à l'aide d'une base de Schauder en dimension réduite, le choix de la base est à la discréption du statisticien, elle peut-être choisie par visualisation simple, par validation croisée avec une méthode des moindres carrées avec les observations, la librairie `scikit-fda` [5] de *Python* m'a permis d'implémenter différents lissages et le lissage par B-spline donne les meilleurs résultats, en pratique, en plus d'apporter une grande flexibilité de part la personnalisation manuelle des noeuds. Une fois les données lissées par une base de fonctions B-splines, on créer une grille d'échantillonnage arbitrairement précise du domaine de définition des observations (par exemple 10 000 points ou 1 000 000 de points) et on reconstruit des observations lisses à l'aide des évaluations des fonctions B-splines en ces points ainsi que les coefficients des observations correspondant à chaque fonction de base.

Soit $n \in \mathbb{N}$ le nombre de points en lesquels on souhaite évaluer les fonctions lissées pour procéder à la convolution.

Soit b le nombre de fonctions B-spline dans la base choisie préalablement.

Soit \mathcal{M} la matrice des coefficients de chaque observation pour chaque fonction de base, ainsi que \mathcal{B} la matrice des fonctions de bases évaluées en les T points arbitraires. On reconstruit le lissage \mathcal{L} en effectuant la multiplication matricielle suivante :

$$\underbrace{\mathcal{M}}_{n \times b} \times \underbrace{\mathcal{B}}_{b \times T} = \underbrace{\mathcal{L}}_{n \times T} \quad (3.1)$$

On applique ensuite de manière tout à fait classique la convolution unidimensionnelle sur la courbe lissée \mathcal{L} en ajustant la taille du filtre, de l'enjambée, ainsi que les autres paramètres de la convolution. Par exemple dans le cadre de la figure 2.4, la donnée d'entrée n'est plus le vecteur de points en bleu, mais la courbe rouge évaluée en un nombre de point arbitrairement grand.

Soient $(k, s, p, d,)$ la taille du filtre, de l'enjambée, du rembourrage, de la dilatation, la sortie TSC de la première couche appliquée à un lot de données \mathcal{X} est alors défini ainsi :

$$TSC(\mathcal{X}) := C_{(k,s,p,d,g)}(\mathcal{L}) = C_{(k,s,p,d,g)}(\mathcal{M}\mathcal{B}) \quad (3.2)$$

avec C définie en l'équation 2.10.

2.1 Avantages de la convolution lissée :

Ce modèle a comme principal avantage de régler les deux problèmes que rencontre la convolution classique appliquée aux données brutes. En effet, peu importe si les observations sont régulièrement mesurées ou pas, ce modèle ne nécessite pas d'ajouter des 0 pour les temps non-observés (qui ont les désavantages énoncés en partie 2 section 1.1), étant donné qu'il effectue un lissage directement à partir des observations. Il peut donc s'appliquer à de plus nombreuses données que le modèle de convolution classique.

De plus il a l'avantage de prendre en entrée des données lisses et prend donc en compte l'hypothèse de structure faite sur les données d'entrée : Le processus à l'origine des observations est un processus naturel qui demande une quantité d'énergie pour varier, et qui par principe ne peut pas avoir des dérivées seconde qui explosent en temps fini, ainsi les processus à l'origine de ces observations doivent être lisse (ou au moins régulier d'ordre 2).

Il est également possible avec un tel modèle d'obtenir des sorties de couche de convolutions qui sont elles des fonctions continues, pas forcément lisse en revanche. Les sorties des couches de convolution après l'entraînement d'un tel modèle peuvent être assez informatives et interprétables à l'instar de la convolution bidimensionnelle en analyse d'image. Le statisticien utilisant un tel modèle a accès au procédé sous-jacent lors de la prise de décision, et la structure fonctionnelle des données est préservée le plus longtemps possible lors de la méthode forward du réseau de neurones.

super

Un autre avantage considérable de la convolution lissée par rapport aux modèles concurrents est la facilité déconcertante avec laquelle on peut inclure les dérivées dans l'analyse. Comme vu dans la section ??, les dérivées des fonctions lissées sont un outil majeur en FDA. Analyser les dérivées d'une variables est analogue à notre intuition d'utiliser la convolution : sur-ligner des caractéristiques locales. Nous avons également évoqué brièvement qu'en analyse d'image, les différentes couleurs de l'images sont traitées comme 3 canaux des trois couleurs primaires rouge bleu et vert, et cette option est également disponible dans les librairies d'apprentissage automatique dans la couche de convolution unidimensionnelle. L'idée du professeur Beaulac a alors été de traiter les dérivées des fonctions lissées comme différents canaux. En effet on peut effectuer la multiplication matricielle 3.1 en remplaçant C la matrice des coefficients des observations par la matrice des coefficients de base des dérivées premières, dérivées seconde, dérivées troisièmes, et traiter tous ces lissages reconstruits comme des canaux, de manière analogue aux canaux de couleurs en analyse d'image. Si l'on devait ajouter ces information dans un modèle MLP cela nécessiterait de lisser les données, les dériver, puis évaluer les dérivées des lissages en le même nombre de points que le nombre de mesures T de base, puis multiplier le nombre de paramètres de la couche d'entrée pour y ajouter les valeurs évaluées des dérivées des lissages, ce qui aurait un coût calculatoire énorme. Notre modèle permet alors de traiter

les dérivées des lissages d'une manière simple et cohérente, c'est à dire que les mêmes noyaux vont apprendre des dérivées jusqu'à un ordre choisi par l'utilisateur. Nous verrons dans la partie 4 que l'ajout des dérivées au modèle permet une augmentation significative des performances de classification sur certains jeux de données.

2.2 Une généralisation de la convolution :

Dans le cadre de données fonctionnelles régulièrement espacées, le modèle classique de convolution 1D est un cas particulier de la convolution lissée ajustable. La propriété suivante est une des propriétés importantes dans notre projet, car elle a permis d'ancrer le modèle de la convolution à une dimension dans un ensemble de modèles plus larges que l'on appelle **modèles de convolution lissée ajustable**.

Propriété : Soit un jeu de données fonctionnelles régulièrement mesurées et

$$\text{espacées } X = \begin{bmatrix} x_1(0) & x_1(t_1) & x_1(t_2) & \cdots & x_1(T) \\ x_2(0) & x_2(t_1) & x_2(t_2) & \cdots & x_2(T) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_p(0) & x_p(t_1) & x_p(t_2) & \cdots & x_p(T) \end{bmatrix}$$

Pour tout modèle de convolution appliquée aux données brutes X , il existe un modèle TSC tel que les deux modèles soient égaux. Autrement dit :

$$\mathcal{H}_{Conv1D} \subseteq \mathcal{H}_{TSC}$$

Démonstration :

Soient les $(x_i), 1 \leq i \leq n$ les observations qui sont toutes mesurées aux même temps $t, 1 \leq t \leq T_0$ où T_0 est le nombre total de temps ou les observations sont mesurées.

Soit n le nombre de points en lequel on évaluera le lissage (notre grille), avec $T_0 \ll n$.

Alors on a sur notre grille régulière $\left[0 : \left\lfloor \frac{T_0}{n} \right\rfloor - 1\right]$, une observation $x(t) \in X$ à une fréquence de $\left\lfloor \frac{n}{T_0-1} \right\rfloor$.

Soit $M_{(k,s,p,d)}$ un modèle d'apprentissage de convolution unidimensionnelle déterminé par H couche cachées, et dont la première couche d'entrée $C1_M$ est déterminée par les hyperparamètres (k, s, p, d) (taille du noyau, stride, padding et dilatation respectivement).

On choisit par exemple \mathcal{L} , comme lissage des données fonctionnelles, l'interpolation par B-spline d'ordre 3. Toutes les observations appartiennent ainsi au lissage :

$$\forall i, 1 \leq i \leq n, \forall t \in [1 : T] (x_i)(t) \in \mathcal{L} \text{ (par définition de l'interpolation)}$$

Soit une couche C de Convolution 1D ayant pour hyperparamètres :

$$\begin{cases} \text{kernel size} = K \\ \text{stride} = \left\lfloor \frac{n}{T_0-1} \right\rfloor \\ \text{padding} = \left\lfloor \frac{n}{T_0-1} \right\rfloor \times 2p \\ \text{dilatation} = \left\lfloor \frac{n-2}{T_0-1} \right\rfloor \end{cases}$$

alors

$$\boxed{\text{C1}_M \left((x_i)_{1 \leq i \leq n} (t)_{t \in [T_0]} \right) = C(\mathcal{L}(x_i))}$$

Les deux couches prennent les mêmes données d'entrée, il ne reste qu'à ajuster les paramètres des couches suivantes du modèle *TSC* pour obtenir le même modèle que le CNN sur les données brutes ce qui conclue la preuve. On peut par exemple retrouver la convolution 1D avec l'exemple des températures comme cas particulier de la convolution lissée ajustable avec les bons paramètres comme sur l'exemple de la figure 3.2

FIGURE 3.2 – Application de la convolution à l’interpolation par B-spline de l’observation météorologique de la figure 2.1. Avec les paramètres décrits, le modèle TSC dont cette couche est la couche d’entrée, appliquée aux données lissées et le modèle de convolution sur les données brutes vu précédemment sont parfaitement égaux car leurs premières couches prennent les même entrées. Le paramètre de dilatation permet un contrôle de la complexité du modèle et rend ajustable la convolution. Cette animation représente bien le fait que TSC soit une généralisation de la convolution dans le cadre des données fonctionnelles.

Cette idée de preuve fonctionne pour tout n mais on note que si l’on prend $n = T_0 + 1$ on peut choisir comme grille d’évaluation la grille régulièrement espacée $[0 : T_0]$, ce qui revient également à appliquer la convolution sur les données brutes et simplifie la preuve. La raison pour laquelle nous introduisons cette idée de preuve avec $n \gg T_0$ est que nous souhaitons montrer la généralité du modèle que nous proposons. Tout d’abord, cette approche fonctionne même si les données sont irrégulièrement espacées. En outre, et c’est là le point de notre contribution, la flexibilité dans le choix d’échantillonnage, de la taille du noyau et de la dilatation et de l’enjambée donne naissance à une multitude de nouveaux modèles formant un spectre intermédiaire entre la convolution discrète et une forme de convolution sur des variables fonctionnelles lisses. Nous avons vu que la convolution discrète est un cas particulier, ou plutôt une sorte de modèle ‘limite’ discret, nous allons à présent étudier le modèle limite à l’opposé

de ce spectre, c'est à dire lorsque notre dilatation et notre enjambée sont les plus petites possibles et que l'on fait glisser un noyau (numériquement) continue sur une courbe (numériquement) continue.

2.3 Continuum entre convolution discrète et continue

On remarque alors que la dilatation joue un rôle de contrôle direct sur la complexité du modèle, car elle nous permet d'ajuster le passage du noyau sur les courbes lissées. Pour évaluer la flexibilité du modèle que nous proposons nous allons étudier les deux extrêmes de ce modèle. Un extrême lorsque l'on choisit une "grande" dilatation et un petit noyau (de taille 5 dans la figure 3.2) on obtient un modèle de convolution discrète unidimensionnelle classique. Nous allons désormais étudier l'autre extrême de ce modèle. Que se passe-t-il si l'on choisit n de plus en plus grand, un noyau définit sur un intervalle, dont la taille augmente avec n la dilatation et l'enjambée la plus petite possible ?

Ce choix de complexité revient à l'analyste utilisateur, quelques règles simples permettent une bonne précision en pratique, comme le fait de choisir les paramètres de la première couche de manière à ce qu'au moins quelques vraies observations soient dans les noyaux à chaque stride de la convolution. Ainsi on évite que le réseau de neurones n'apprenne sur trop de *d'interpolation*, c'est à dire des observations créées de toute pièces par le lissage. On note également qu'avec une dilatation de 1 et un stride de 1, les noyaux de la première couche de convolution coulisse de plus en plus continuellement sur les courbes lisses, à mesure que n augmente, approchant ainsi numériquement la corrélation croisée continue entre un noyau continu et le lissage.

Soient $\mathcal{X} := \left(x_i(t)_{t \in [T]} \right)_{1 \leq i \leq p}$ nos observations, $\mathcal{L}_{\mathcal{X}} := (\mathcal{L}_{x_i}(t)_{t \in \sigma_{\mathcal{L}}})_{1 \leq i \leq p}$

leurs lissages, définis sur $\sigma_{\mathcal{L}} := (0 = t_0, t_1, t_2, \dots, t_n = T_0)$, subdivision régulière de $[0, T_0]$ avec un pas de $\frac{1}{n}$. Puisque $\sigma_{\mathcal{L}}$ comporte n termes, elle est en bijection avec $[n]$ et on utilisera souvent ce fait dans cette partie. Soit $T_{\mathcal{K}} \in [T_0]$ et supposons que l'on ait un noyau \mathcal{K} défini sur une subdivision $\sigma_{\mathcal{K}} := (0 = t_0, t_1, t_2, \dots, t_K = T_{\mathcal{K}})$ de l'intervalle réel $[0, T_{\mathcal{K}}] \subseteq [0, T_0]$. On voit $\sigma_{\mathcal{K}}$ comportant $\left\lfloor n \frac{T_{\mathcal{K}}}{T_0} \right\rfloor$ termes comme une partie de la subdivision σ_{T_0} qui elle comporte n termes. Pour des raisons de lisibilité on notera parfois également $\mathcal{R}_{\mathcal{K}} := \frac{T_{\mathcal{K}}}{T_0}$, ce ratio représente la part de l'ensemble de départ qui est couverte par le domaine de définition du noyau \mathcal{K} .

Choisissons comme noyau de convolution le noyau suivant

$$\mathcal{K}_n : \sigma_{\mathcal{K}} \mapsto \mathbb{R}$$

$$t_k \longmapsto \frac{1}{n} \mathcal{K}(t_k)$$

de taille $K = \left\lfloor n \frac{T_{\mathcal{K}}}{T_0} \right\rfloor = \lfloor n \mathcal{R}_{\mathcal{K}} \rfloor$.

On choisit alors la dilatation et l'enjambée les plus petites possibles, c'est-à-dire $d = 1$, $s = 1$ et pas de rembourrage ($p = 0$). De part l'équation (??) la taille de la donnée de sortie s'écrit :

$$T_1 = n - (K - 1) + 1 = n - \left\lfloor n \frac{T_K}{T_0} \right\rfloor + 2 = n \frac{(T_0 - T_K)}{T_0} + cste = n(1 - \mathcal{R}_K) + cste$$

Soit la subdivision $\sigma_{T_1} = \left[0, \frac{1}{T_1}, \frac{2}{T_1}, \dots, \frac{T_0}{T_1}\right]$ de l'intervalle réel $[0, T_0]$. Puisque les indices des sorties de la convolution sont arbitraires, on peut arbitrairement décider, puisque σ_{T_1} est en bijection avec $[T_1]$, de définir la convolution entre \mathcal{L} et \mathcal{K} sur σ_{T_1} , qui est une subdivision dont le pas est

$$\frac{1}{T_1} = \frac{1}{n(1 - \mathcal{R}_K)} + cste \xrightarrow[n \rightarrow +\infty]{} 0$$

Lorsque n devient arbitrairement grand, la subdivision σ_{T_1} va donc s'approcher de l'intervalle réel $[0, T_0]$. Maintenant que nous avons le domaine de définition de l'opération de convolution entre le lissage \mathcal{L} et le noyau \mathcal{K}_G , explicitons cette opération comme définie dans l'équation (2.10) :

$$\forall s \in \sigma_{T_1} :$$

$$\begin{aligned} TSC_n(\mathcal{X})(s) &= C(\mathcal{L}_{\mathcal{X}})(s) = \sum_{k=0}^K \mathcal{K}_n(t_k) \mathcal{L}(s + t_k) \\ &= \sum_{k=0}^{\lfloor n\mathcal{R}_K \rfloor} \frac{\mathcal{K}(t_k) \mathcal{L}(s + t_k)}{n} \\ \text{puisque } \forall k \in [0 : K - 1], \quad &\begin{cases} t_k \in [0, T_K] \\ \frac{1}{n} = t_{k+1} - t_k \\ \lim_{G \rightarrow +\infty} \left\lfloor n \frac{T_K}{T_0} \right\rfloor = +\infty \end{cases} \quad \text{on a :} \end{aligned}$$

$$TSC_n(\mathcal{X})(s) = \sum_{k=0}^{\lfloor n\mathcal{R}_K \rfloor} \mathcal{K}(t_k) \mathcal{L}(s + t_k) \underbrace{(t_{k+1} - t_k)}_{\substack{\longmapsto 0 \\ n \rightarrow +\infty}} \xrightarrow{n \rightarrow +\infty} \int_0^{T_K} \mathcal{K}(t) \mathcal{L}(s + t) dt$$

D'une part, la fonction

$$\Phi_{T_K} : s \in \mathbb{R} \mapsto \int_0^{T_K} \mathcal{K}(t) \mathcal{L}(s + t) dt$$

n'est autre que la convolée Φ entre le noyau et le lissage comme définie en équation 2.7. D'autre part $\lim_{n \rightarrow +\infty} T_1 = +\infty$, ainsi la subdivision σ_{T_1} s'approche de

l'intervalle continue $[0, T_0]$ à mesure que n grandit. La fonction définie par TSC_n ayant pour support σ_{T_1} s'approche d'une fonction définie sur l'intervalle $[0, T_0]$ tout entier lorsque n devient arbitrairement grand. Ainsi le modèle limite de ce que nous proposons lorsque la grille d'évaluation s'approche de la droite réelle est un modèle de convolution continue, telle que définie en théorie de l'intégration, dans lequel la grille d'évaluation du noyau \mathcal{K} , pouvant devenir arbitrairement grande elle aussi, est apprise par le réseau de neurones afin de discriminer lissages des observations. Il semble que le modèle limite soit trop complexe et trop couteux en calcul, ce qui signifie d'un point de vue de la décomposition de l'erreur 2.6

FIGURE 3.3 – Convolution lissée ajustable avec $T_0 = 12$, $T_{\mathcal{K}} = 1$, $n = 10^5$ stride=1, dilatation=1, on simule une subdivision proche de 0 (autrement dit $n \mapsto +\infty$) appliquée à l'interpolation de nos données de température vue en figure 3.2.

La convolée entre le lissage et le noyau apparaît continue à l'oeil nu, et cette opération s'approche de la convolution comme elle est définie en théorie de l'intégration avec la mesure de Lebesgue. Le noyau est ici définie quasi-continûment sur l'intervalle $[0, 1]$ subdivisé en $K = \left\lfloor \frac{10^8}{12} \right\rfloor$ termes, et ainsi la donnée fonctionnelle de sortie a pour support l'intervalle $[0 : 12]$ découpé en T_1 termes où par l'équation 2.9 :

$$T_1 = 10^5 \times \left(1 - \frac{1}{12}\right) \approx 9.17 \times 10^4$$

ce qui à l'oeil nu est indiscernable de la continuité.

Le modèle que nous proposons est alors situé sur un spectre avec d'un côté la convolution unidimensionnelle, et de l'autre, ce qui s'apparente à la convolution continue entre les filtres et les variables aléatoires fonctionnelles de dimension

infinie. Nous avons vu dans cette partie comment K la taille du noyau, le pas s , et surtout la dilatation d jouent un rôle de contrôle de la complexité du modèle. Le modèle convolutif unidimensionnel étant sans doute trop "simple", et le modèle continu sans doute trop complexe.

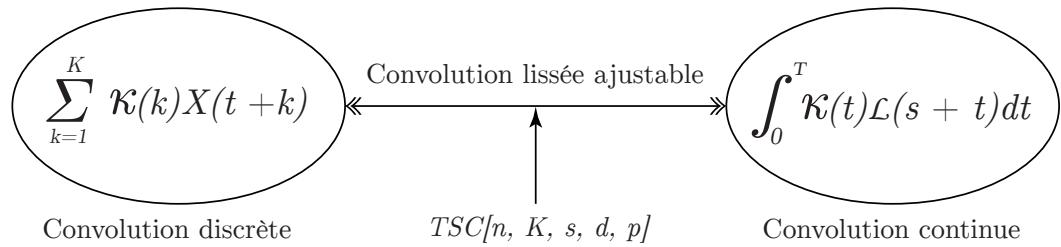


FIGURE 3.4 – Continuum entre les deux convolutions. Il est possible d'ajuster tous les paramètres de la convolution lissée ajustable pour se placer sur ce continuum. Le nombre de modèles possible étant extrêmement grand, la sélection par validation croisée serait beaucoup trop couteuse et des choix basés sur les intuitions et ou une expertise en statistique seront nécessaire pour les ajuster.

CHAPITRE 4

EXPÉRIMENTATIONS

2 Méthode d'expérimentation

2.1 Jeux de données

Pour évaluer les modèles et les comparer nous allons utiliser quatre jeux de données classiques en classification fonctionnelles .

- **Phoneme** : Le jeu de données phonème comprend des enregistrement vocaux de phonèmes et la tâche à effectuer est de classifier automatiquement les enregistrement vocaux dans les 5 catégories de voyelles a, e,i, o ,u. On a accès pour cela à 500 observations d'estimations spectrales par log-périodogramme discrétilisés de fréquences vocales.
- **El Nino** : Températures dans l'air mesurées mensuellement au niveau de la mer dans 4 régions du golf du Mexique. On a accès à 148 observations que l'on cherche à classer les données selon les 4 régions.
- **SOFA** : Scores SOFA (Sequential Organ Failure Assessment) de 520 patients au cours d'une étude de survie. La variable d'intérêt est la mort du patient. On coupe l'étude à partir de 20 jours, et on effectue de la classification binaire pour tenter de prédire si le patient va mourir ou non au cours de l'étude. Les données sont censurées par intervalle donc elles sont irrégulièrement mesurées et espacées. Pour pouvoir appliquer le MLP on ajoute artificiellement des 0 dans les données pour que toutes les observations aient le même support de définition, ce qui n'est pas nécessaire pour notre modèle TSC ou pour les réseaux de neurones récurrents.
- **Canadian Weather** : Températures et pluviométries mesurées quotidiennement dans 35 villes du Canada à classer dans 4 régions différentes. Les deux courbes de données (température et pluviométrie) doivent être analysées indépendamment par tous les modèles excepté TSC qui voit ces deux

courbes comme deux canaux d'une même "image" fonctionnelle. Ce jeu de données est le plus petit en terme de nombre d'observations.

2.2 Évaluation des performances :

Afin d'évaluer les performances de classification des modèles on commence par séparer l'ensemble des données en ensemble d'entraînement et ensemble de test. Nous comparons les résultats de classification des modèles appliqués aux données de test aux classes de test réelle, et calculons un pourcentage. Puisque l'initialisation des modèles ainsi que l'entraînement comportent tous deux des parts d'aléatoire, nous avons choisi d'utiliser une méthode de Monte-Carlo, dans laquelle on effectue des simulations d'initialisations puis d'entraînements de modèles un grand nombre de fois, afin d'éliminer l'aspect aléatoire de l'initialisation des poids des réseaux de neurones, ainsi que le caractère aléatoire de la descente de gradient stochastique. Dans la suite, on effectue 50 simulations de chaque modèle et sur une centaine d'époques d'entraînement. On mesure alors le nombre moyen de bien classé par le modèle dans l'ensemble de test au fur et à mesure de l'entraînement et on trace les courbes d'entraînement. On effectue une moyenne sur le nombre total de simulation et traçons également les intervalles de confiance au niveau 95%. Il est également possible d'effectuer des tests de student pour se faire une idée de la significativité statistique de l'avantage de précision qu'un modèle a sur un autre.

2 Choix de l'architecture :

Nous allons nous intéresser à la tâche de classification de données fonctionnelles. Puisque la couche d'entrée TSC comporte de nombreux nouveaux paramètres comme les noeuds du lissage, le nombre de points d'évaluation de ce lissage, la base choisie ainsi que le degré des fonctions B-spline, ce modèle peut s'avérer particulièrement dur à paramétrier. La règle de pouce que nous utilisons est de prendre une évaluation de 7500 points et de choisir un petit nombre de noyaux, avec un pas assez grand, *i.e.* une dilatation de 75, 8 noyaux de taille 7 ainsi qu'une enjambée assez grande (de 100) pour éviter que le calcul ne soit trop lent. En pratique, s'aventurer dans des valeurs trop grandes pour le nombre n et une enjambée et dilatations très faible peuvent mettre hors de mémoire les processeurs graphiques, ou donner lieu à des heures de calcul. C'est pourquoi il est important de composer avec ses ressources en calcul. Je suis personnellement muni de CUDA ce qui me permet de faire porter les modèles et leurs multiples entraînements par mon unique GPU, mais si l'on dispose d'une puissance de calcul plus importante ainsi qu'une plus grande mémoire vive, alors il est envisageable de prendre plus de points d'évaluation des fonctions lissées. Les valeurs que j'ai énoncé donnent en moyenne les meilleures performances que j'ai pu trouver en un temps raisonnables, mais il est envisageable d'adapter absolument toutes

les valeurs pour maximiser la performance, et la liste des tests à effectuer est immense, il est sûrement possible de faire bien mieux. La fonction de perte que nous cherchons à maximiser est l'entropie croisée car nous effectuons de la classification [2.4](#), à noter qu'il est également possible de choisir l'écart quadratique normalisée ou MSE telle qu'elle est définie dans l'équation [2.5](#), qui donne des résultats très similaires, sauf pour **SoFA** pour laquelle les résultats sont légèrement meilleurs avec l'entropie croisée. Nous minimisons cette perte en remontant son gradient avec l'algorithme [3.1](#), et un pas d'apprentissage = 0.00089 car c'est celui qui fonctionne le mieux. Nous allons commencer par optimiser l'architecture du réseau de neurones TSC pour pouvoir le comparer ensuite aux modèles qui ont déjà été développés par le passé.

2.1 Nombre de couches :

Choisir le nombre de couche de convolution ne semble pas évident car les interactions entre les couches semblent chaotiques et non-linéaires. Créer un algorithme afin de faire ce choix serait complexe et relève de l'ingénierie plus que de la recherche en statistiques. C'est pourquoi nous nous sommes concentrés sur quelques tests et avons déduit que 3 couches semblaient être le nombre optimal, en rajouter rallonge le calcul sans augmenter les performances et en enlever heurte les performances du modèles. La figure [4.1](#) compare un modèle à une couche contre le modèle optimal. Nous conserverons trois couches dans la suite de notre travail.

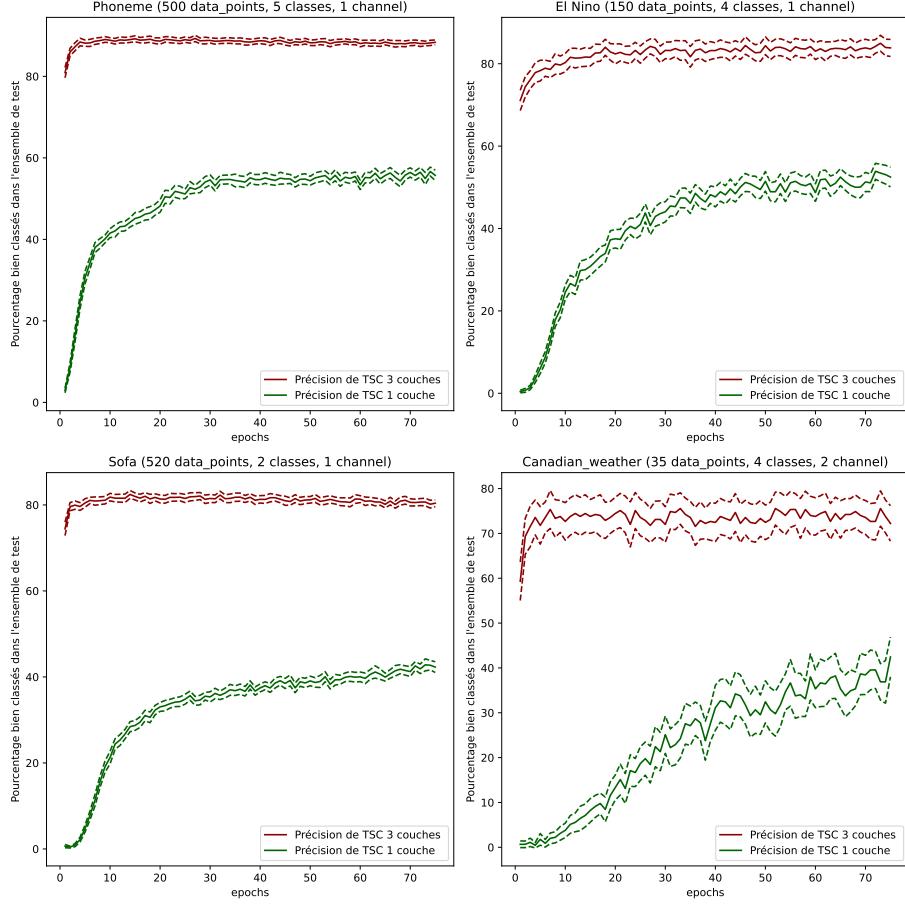


FIGURE 4.1 – TSC 3 couches vs TSC 1 couche. Bien que le choix de couches de convolution semble arbitraire, le modèle à une couche est bien moins rapide en apprentissage que le modèle à trois couches, qui donne les meilleurs résultats en pratique.

2.2 Couches non-paramétriques :

Choix du redresseur :

La fonction ReLU (Rectified Linear Unit) est souvent utilisée dans les réseaux de neurones, elle est définie comme $ReLU(x) = \max(x, 0)$ et sert à augmenter la performance des réseaux de neurones en pratique comme démontré par Talathi *et al* [12]. Au lieu d'annuler les valeurs négatives, dans le cadre de notre modèle, il est plus efficace de les atténuer avec une pente négative $\alpha > 0$. C'est ce que

fait la couche Leaky Relu définie ainsi :

$$\text{LeakyRelu} : x \mapsto \begin{cases} x, & \text{si } x \geq 0 \\ \alpha x, & \text{si } x \leq 0 \end{cases}$$

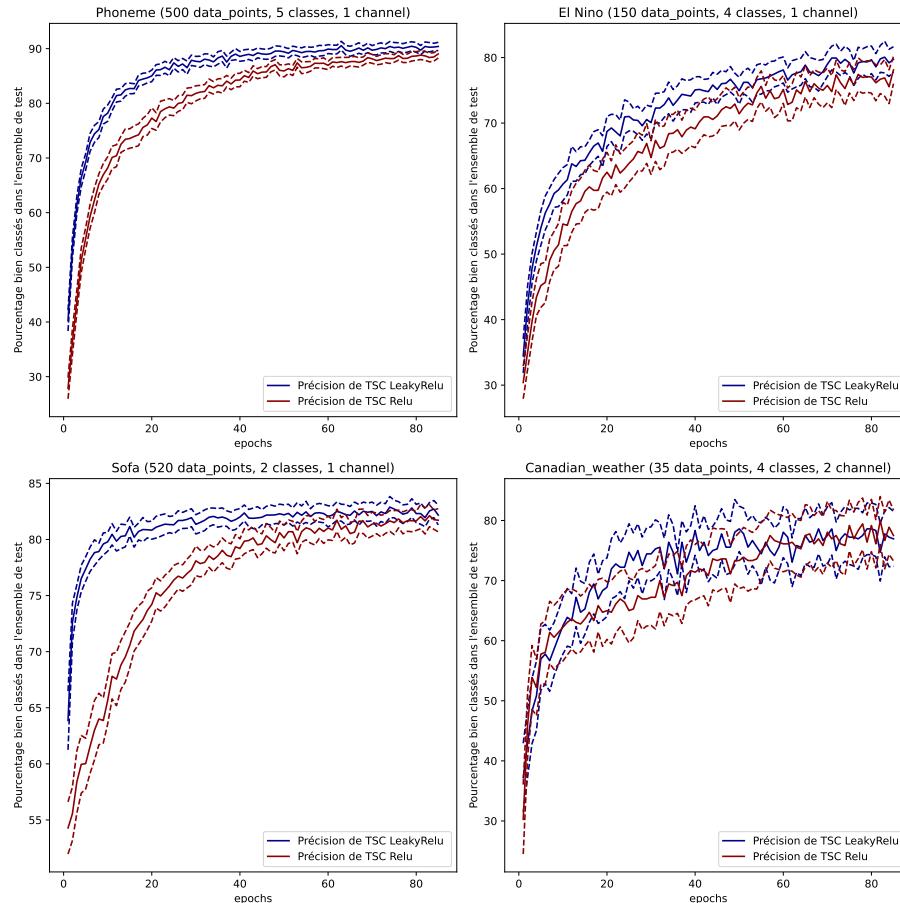


FIGURE 4.2 – TSC avec 3 couches RELU contre TSC avec 3 couches Leaky RELU (avec pente négative de 0.17), La couche Leaky Relu dans chacune des couches de la convolution apporte un avantage significatif pour l'apprentissage, et nous conservons cette couche au lieu de la couche ReLU classique dans la suite.

Cette couche donne de meilleur résultats empiriquement comme le montre la figure 4.2. On ajoutera alors par la suite des couches LeakyRelu avec une pente choisie arbitrairement (0.17 donnant de bons résultats en pratique) entre chaque couches de la convolution et dans la couche dense finale.

Normalisation non-paramétrique :

Une couche très importante dans les réseaux de neurones est la couche non paramétrique de normalisation par lot (ou Batch Normalization). L'article *How does Batchnormalization help optimization ?* [8] démontre ce phénomène empiriquement en apportant des justifications théoriques. Il en va de même dans le cadre de notre modèle, la normalisation par lot est nécessaire pour obtenir des résultats satisfaisant en moins d'une centaine d'époques d'entraînement 4.3. La couche estime la moyenne μ_B et l'écart type σ_B de la sortie de la couche qui arrive avant elle, puis elle retire la moyenne et divise par l'écart type : $BN(x_B) = \frac{x_B - \mu_B}{\sigma_B}$

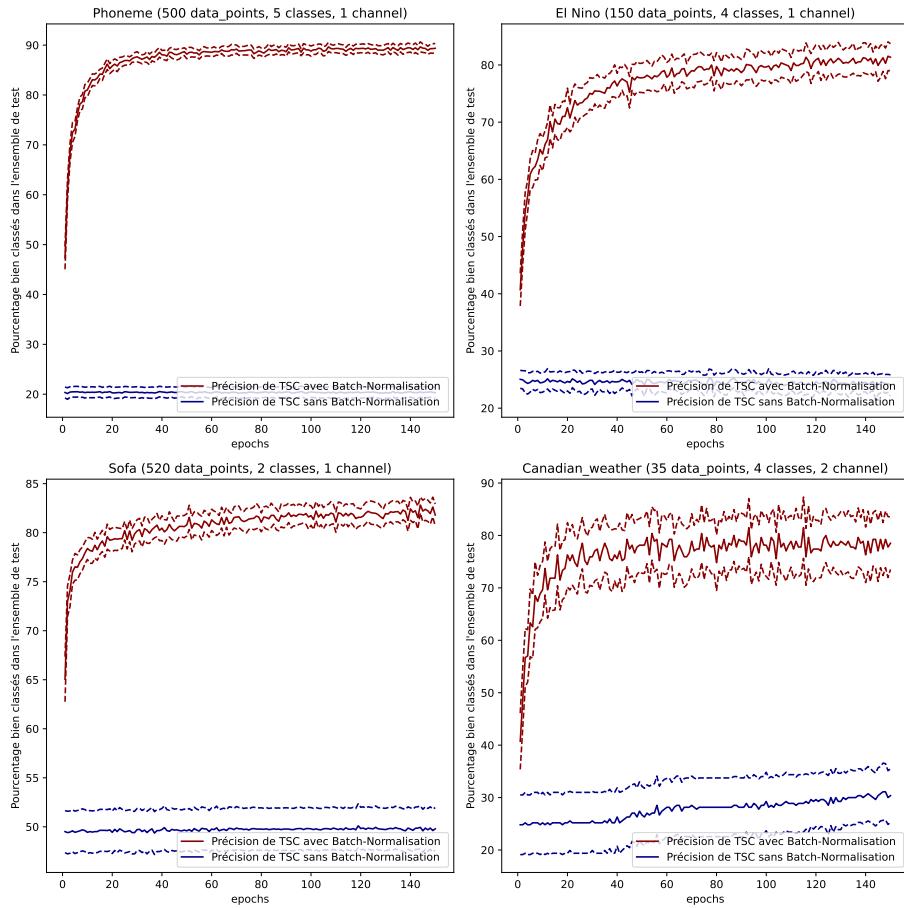


FIGURE 4.3 – Normalisation par lot sans Normalisation par lot, La normalisation des sorties dans chacune des couches de convolution a un impact majeur sur la vitesse de l'apprentissage. Ce graphique justifie l'utilisation de la normalisation de lot dans chacun de nos modèles pour la suite.

2.3 Choix du regroupement (Pooling) :

Nous comparons alors le modèle sélectionné précédemment sans les couches de regroupement avec celui dans lequel on ajoute une couche de regroupement par maximum (MaxPooling) comme c'est souvent le cas dans la littérature en analyse d'image (comme c'est le cas dans [4], [2] par exemple). C'est alors naturellement que l'ajout de cette couche non-paramétrique se présente à nous, et les tests semblent montrer que les performances sont améliorées par l'ajout de ces couches.

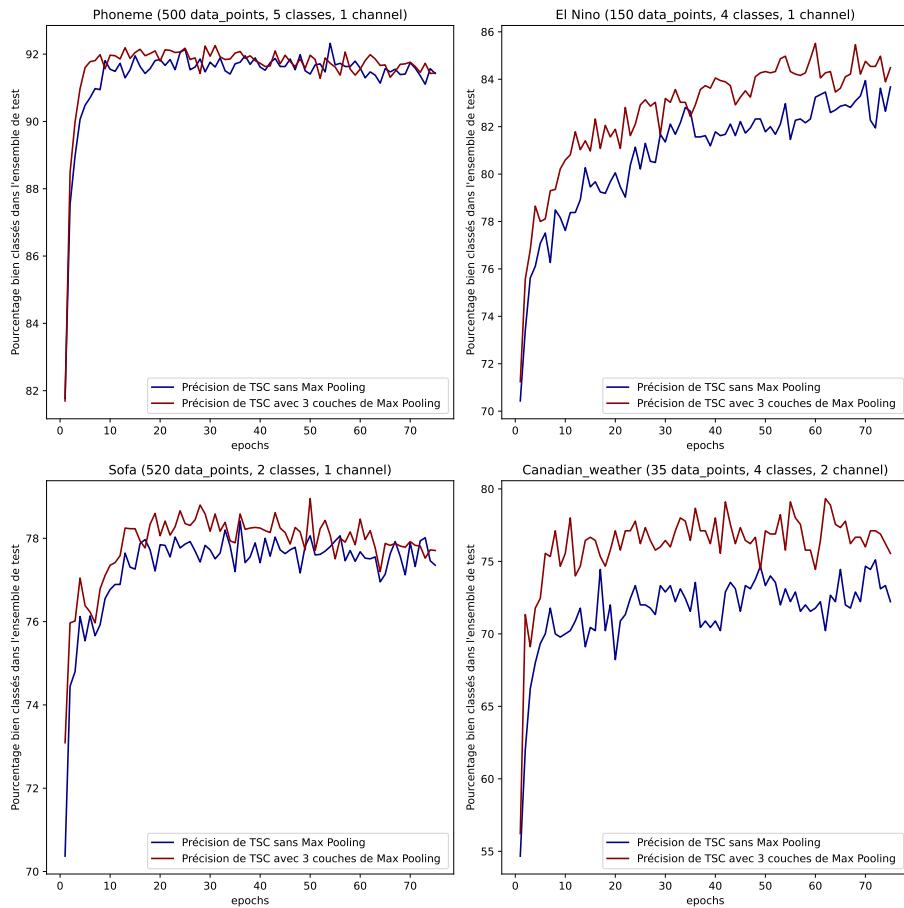


FIGURE 4.4 – TSC avec 3 couches de Maxpooling (à chaque sortie de couche de convolution comme dans la figure 3.1 contre le réseau de neurones sans ces couches. Les résultats semblent être meilleur visuellement cependant les tests ne donnent pas d'avantage significatif à par pour Canadian Weather.

Il existe également une autre forme de regroupement, qui est le regroupement L_p , avec p laissé en option à l'utilisateur. Cette couche calcule la norme L_p des

objets d'entrée pris dans le noyau, en glissant le noyau de la même manière que dans les figures 2.6 et 3.1. Nous avons alors testé les couches de regroupement L_p avec $p = 1$ et $p = 2$, contre le Max Pooling. Il est à noter que le Max Pooling est un cas particulier de la couche Lp Pooling avec $p = \infty$. C'est cependant la norme sup, donc le MaxPooling qui remporte ce test et nous conserverons cette couche par la suite pour la comparaison avec d'autres modèles.

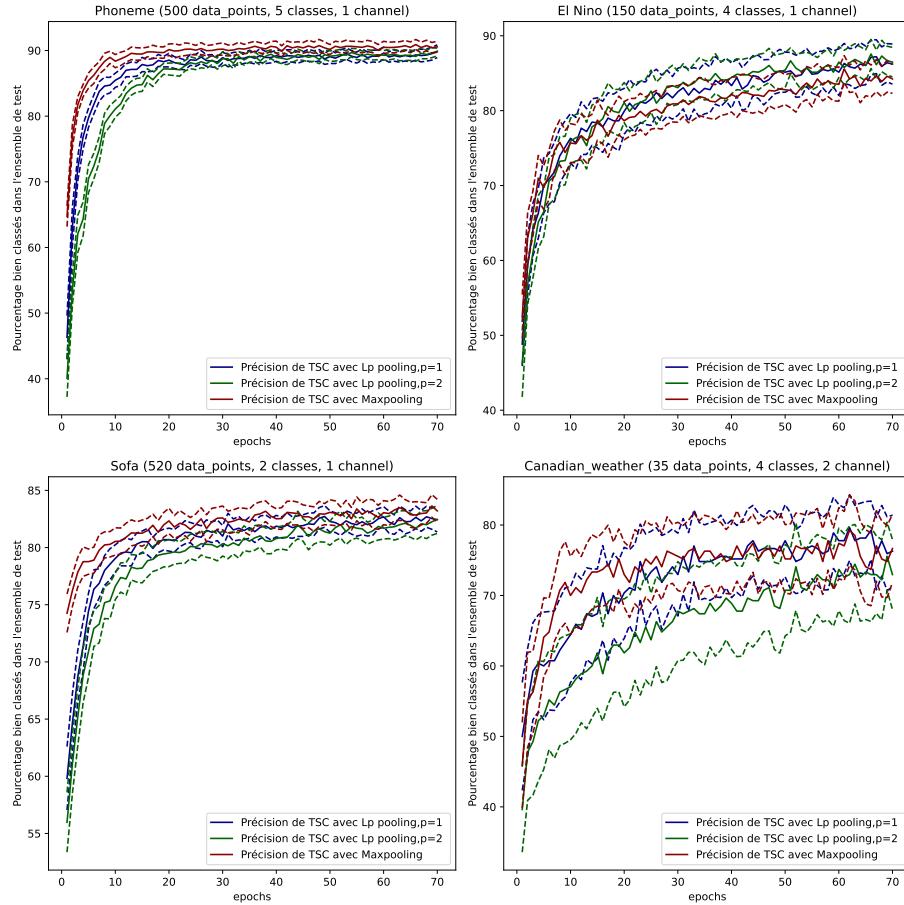


FIGURE 4.5 – Maxpooling vs Lp Pooling avec $p=1$ et $p=2$. Dans la plupart des cas le Maxpooling ($p = +\infty$) donne de meilleures performances que les autres formes de pooling.

2.4 Paramètres propres à TSC :

Choix du lissage : Puisque le modèle va apprendre des lissages des observations et non des observations brutes, choisir le lissage est une des étapes les plus cruciales du modèle. Nous allons pour cela déterminer les deux composantes qui déterminent le lissage : les noeuds, ainsi que le degré. Nous allons commencer par fixer les

autres paramètres, et évaluer les performances de différents modèles TSC avec différents lissages, en commençant par le choix des noeuds, puis nous choisirons l'ordre des fonctions B-splines. En pratique le nombre de noeuds variant de 4 à 6 donne les meilleurs résultats, et pour des raisons pratiques on les choisit régulièrement espacés sur les domaines de définition des données. En ce qui concerne le degré des fonctions de base degrés supérieur à 5 donnent des lissages dont les dérivées et dérivées seconde sont trop grandes par rapport à la forme générale des données, et le degré 1 crée juste une moyenne mobile des données. C'est la raison pour laquelle nos tests se concentreront sur les ordres 2,3,4 et 5, ce qui est certes arbitraire, mais comme précisé au début de cette section 4, les tests à effectuer sont sans fin et des choix s'imposent. La pratique tend à confirmer le choix de ces degrés, et de même pour le nombre de noeuds.

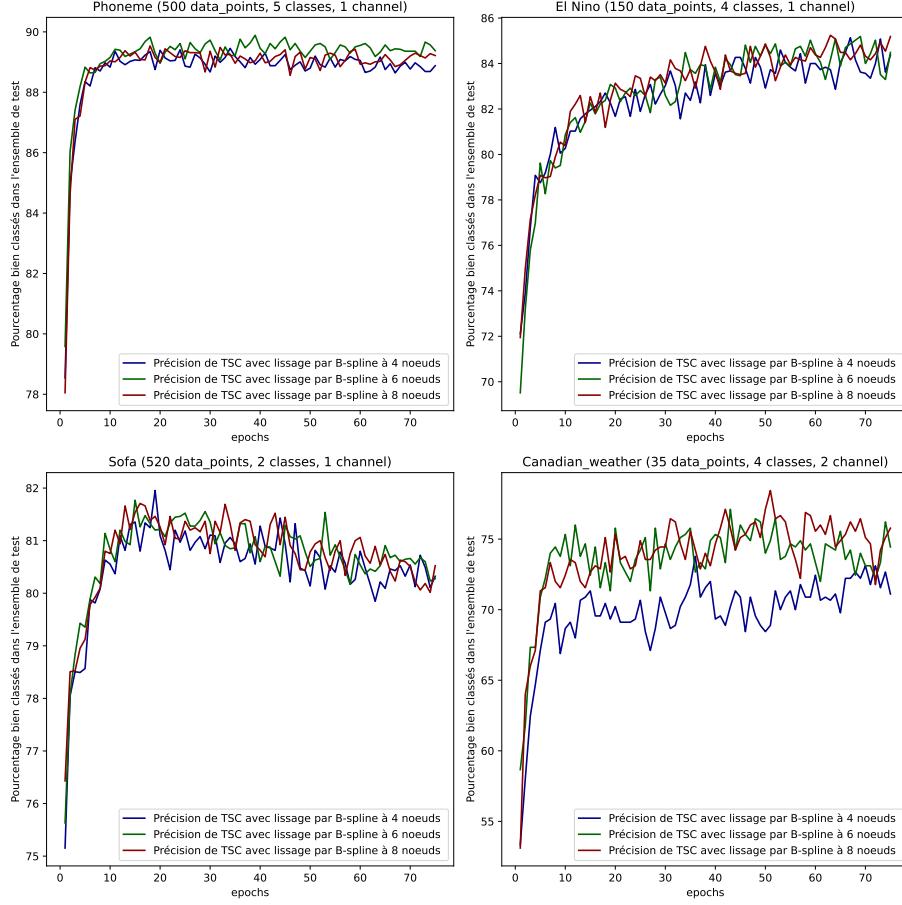


FIGURE 4.6 – Choix du nombre de noeuds pour le lissage, tous les lissages sont d’ordre 4 (arbitraire), 4 semble être moins bon que les autres sur Canadian Weather, en dehors de cela rien de significatif, le choix des noeuds semble arbitraire mais à l’avenir nous choisirons 6 noeuds car la matrice des coefficients sera de plus petite dimension, ce qui réduit (très légèrement) le temps de calcul. On voit que sur les données **SOFA**, on a un peu de sur-apprentissage (overfit), c’est-à-dire que la qualité des prédictions diminue après un certain nombre d’époques, les modèle collent trop aux données d’entraînement, c’est un phénomène lié au fait que les données SOFA nous n’analysons que le score sofa et pas les autres variables du jeu de données, et que le nombre de patients d’entraînement est relativement grand.

Après avoir fixé le nombre de noeuds des fonctions de base passons au choix des degré des lissages. Nous comparons les ordres les plus utilisés en pratique d’après Ramsay et Silvermann (2005) [6], c’est à dire de 2 à 5. Aucune différence

significative n'est relevable. Cependant, l'ordre 4 semble meilleur pour **SOFA** et l'ordre 3 pour le reste. Ce sont donc les paramètres que nous conserverons par la suite.

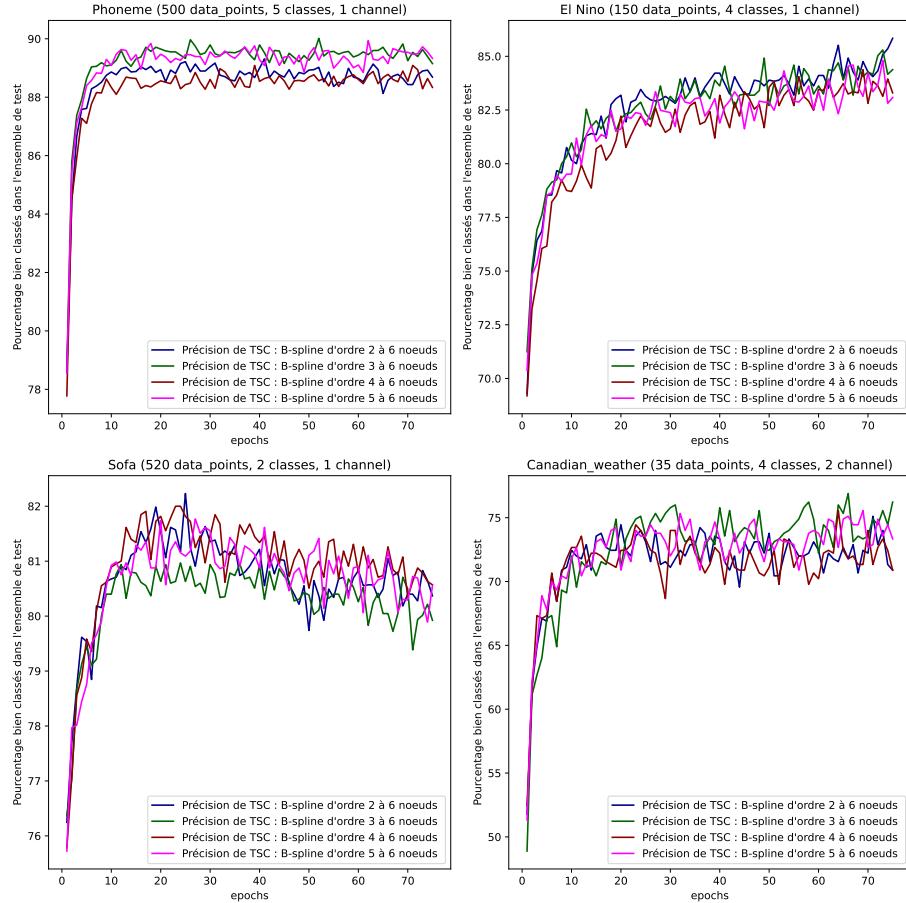


FIGURE 4.7 – Choix de l'ordre des lissages, tous les lissages sont d'ordre 4 (arbitraire), 4 semble être moins bon que les autres sur Canadian Weather, en dehors de cela rien de significatif, le choix des noeuds semble arbitraire mais à l'avenir nous choisirons 6 noeuds car la matrice des coefficients sera de plus petite dimension, ce qui réduit (très légèrement) le temps de calcul.

Comme nous l'avons vu en partie 1.3, l'ajout des dérivées aux modèles d'analyse de données fonctionnelles est courant et apporte souvent une information supplémentaire à l'analyse. Puisque les données sont lissées par une expansion de B-spline avant d'être entrée dans le réseau de neurones, les entrées du réseau de neurones admettent des dérivées et on peut évaluer les dérivées sur le même domaine de définition que celles-ci. De plus comme nous utilisons la couche de convolution implémentée dans Pytorch, nous pouvons utiliser les dérivées évaluées sur le domaine de définition des données comme différents canaux. Cela va améliorer légèrement nos performances selon le jeu de données. La qualité de prédiction sur le jeu de données **El Nino** bénéficient le mieux de l'ajout de ces dérivées. Il est assez difficile de comprendre le phénomène derrière cette amélioration, mais on peut imaginer que les variations dans le jeu de données **El Nino** sont plus caractéristiques de leurs classes respectives. La classification sur les données **SoFA** bénéficient également de l'ajout de ces dérivées de façon statistiquement significative.

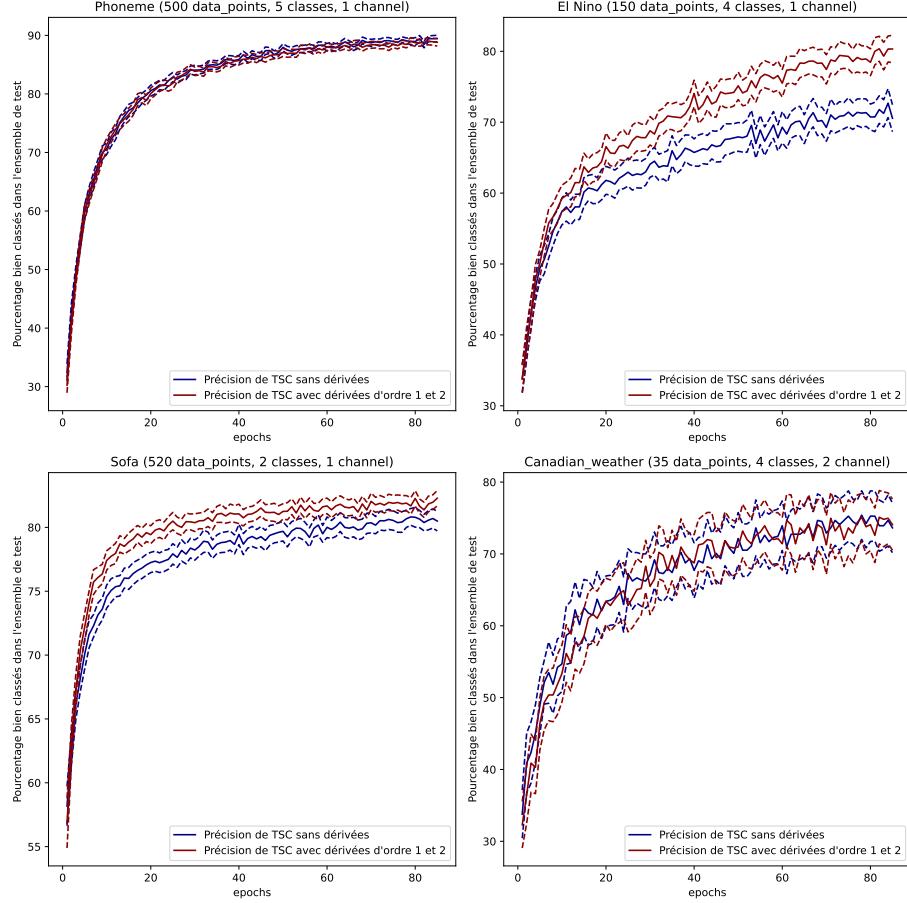


FIGURE 4.8 – TSC avec dérivées d’ordre 1 et 2 contre l’analyse TSC de la fonction seule. Le modèle incorporant les dérivées comme différents canaux de la donnée d’entrée est toujours au moins meilleur que celui n’ayant accès qu’à la fonction seule.

3 Comparaison à d’autres modèles

3.1 Convolution discrète classique vs TSC

On choisit ici de paramétriser TSC de manière à imiter la convolution 1D mais en prenant une observation appartenant au lissage à mi-chemin entre deux réelles observations, la performance est grandement améliorée. En effet sous l’hypothèse que le lissage est une représentation cohérente du processus ayant généré les données observées, on donne juste deux fois plus de données au modèle pour s’entraîner. C’est comme si on avait pris deux fois plus de mesures et peu importe le modèle, le nombre et la qualité des données prévalent toujours dans les tâches

d'apprentissage. On note que si le lissage était aberrant et ou mauvais, les performances du modèle pourraient grandement en pâtrir.

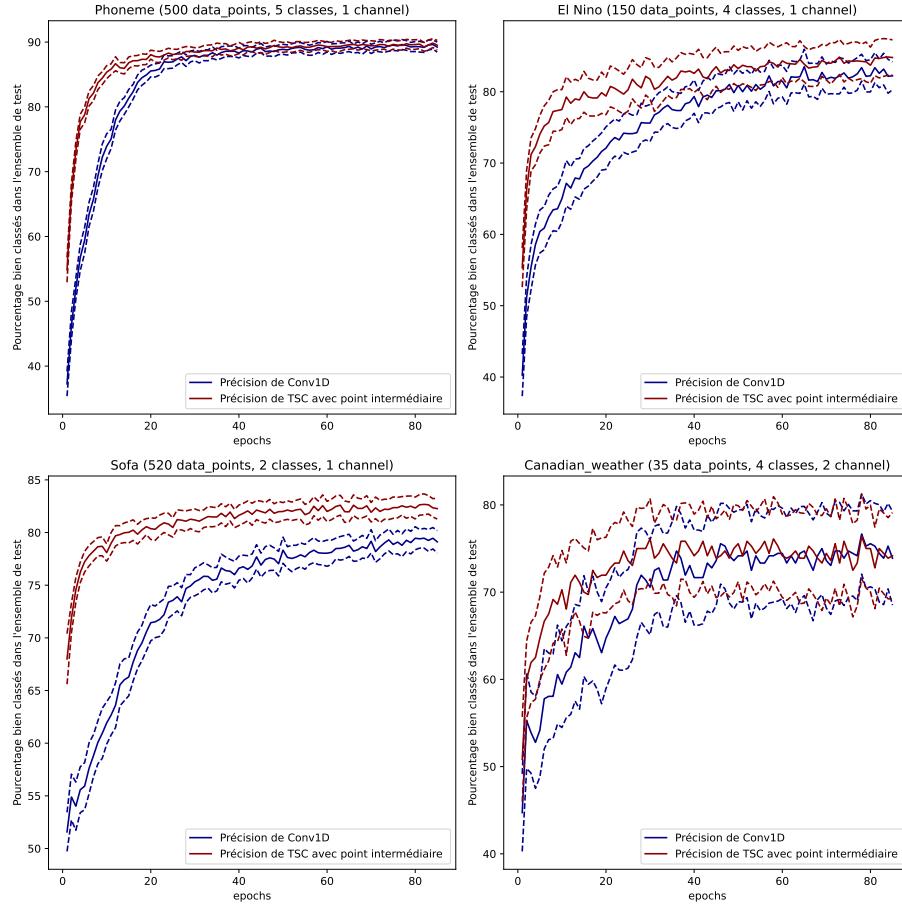


FIGURE 4.9 – Convolution unidimensionnelle contre TSC avec un point intermédiaire, amélioration significative de la vitesse à laquelle le modèle est performant sur l'ensemble de validation pour 3 des jeux de données. Même sur les données de Canadian weather, la précision semble légèrement meilleure au départ.

La figure 4.9 est une illustration parfaite de la contribution que nous avons apporté. Ce graphe nous permet de nous convaincre qu'en augmentant la complexité de la classe \mathcal{H} du modèle de la convolution, nous avons donné naissance à d'autres types de modèles plus rapidement performant que celui dont le biais est trop grand. Nous espérons que d'autres praticiens, chercheurs ou ingénieurs s'intéresseront à notre travail à l'avenir pour trouver des modèles encore meilleurs en faisant varier les multiples hyper-paramètres qui régissent les modèles à convolution lissée ajustable. Maintenant que nous avons trouvé des paramètres paraissant

suffisamment bons, nous allons passer à l'épreuve du feu : la comparaison avec les modèles existant.

3.2 Comparaison générale

Maintenant que les paramètres et l'architecture de notre modèle est choisi, nous pouvons procéder à la comparaison avec d'autre modèles. L'état de l'art actuel d'apprentissage profond en FDA utilise surtout 3 types de modèles : Les modèles de réseau de neurones récurrents dont sont un exemple GRU (Gate Recurrent Unit) et LSTM (Long-Short Term Memory), les réseaux de neurones denses ou complètement connecté (MLP) et les réseaux de neurones de convolution unidimensionnel, dont notre modèles TSC est une généralisation. Nous ne nous attarderons pas sur le fonctionnement des réseaux de neurones récurrents car là n'est pas l'objectif de notre travail, cependant ils sont très bien décrit dans l'article *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network* (A. Sherstinsky (2020)).[10] Les figures 4.10 et 4.11 sont les mêmes figures au détail près de l'intervalle de confiance qui peut venir réduire la visibilité mais exposer les avantages significatifs des modèles aux autres de manière visuelle.

Remarque : Il est à noter qu'un des avantages de la convolution comme opération est le fait de pouvoir paralléliser les calculs sur de multiples processeurs, si l'on dispose de multiples outils de calcul et que les sont données lourdes, avantage qui n'est ni offert ni par le MLP ni par les RNN. Cela peut ne pas être utile dans le cadre de données qui sont peut nombreuses, et peu lourdes mais même si la convolution lissée ajustable est un peu plus lente à exécuter en moyenne que ses concurrents sur les données que nous avons choisi, son coût en calcule va rester relativement constant peu importe la dimension des données, si les paramètres de lissage sont gardés tel qu'ils sont définis ici (ce qui correspondrait à une réduction de dimension si par exemple on avait 100 000 mesures par observation).

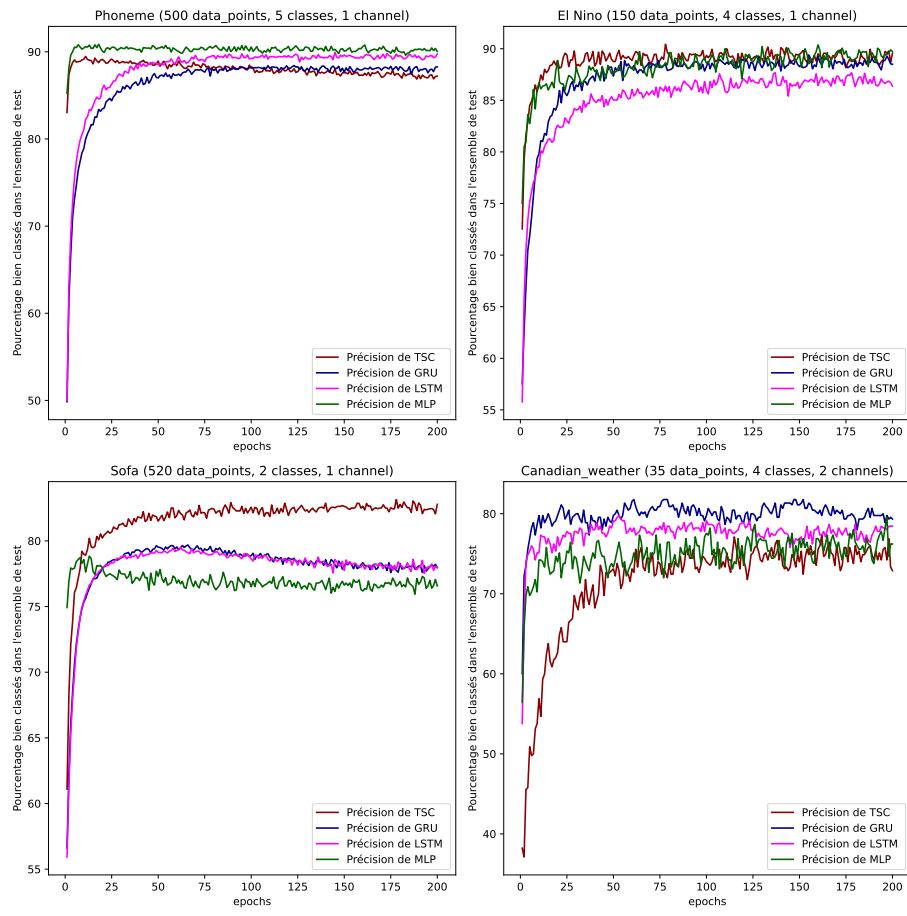


FIGURE 4.10 – TSC comparée à Long-short Term Memory, Gate Recurrent Unit et MLP sans intervalles de confiance. Notre modèle est meilleur sur les données SOFA, et cela est sûrement du au fait que ces données sont censurées donc irrégulièrement espacées.

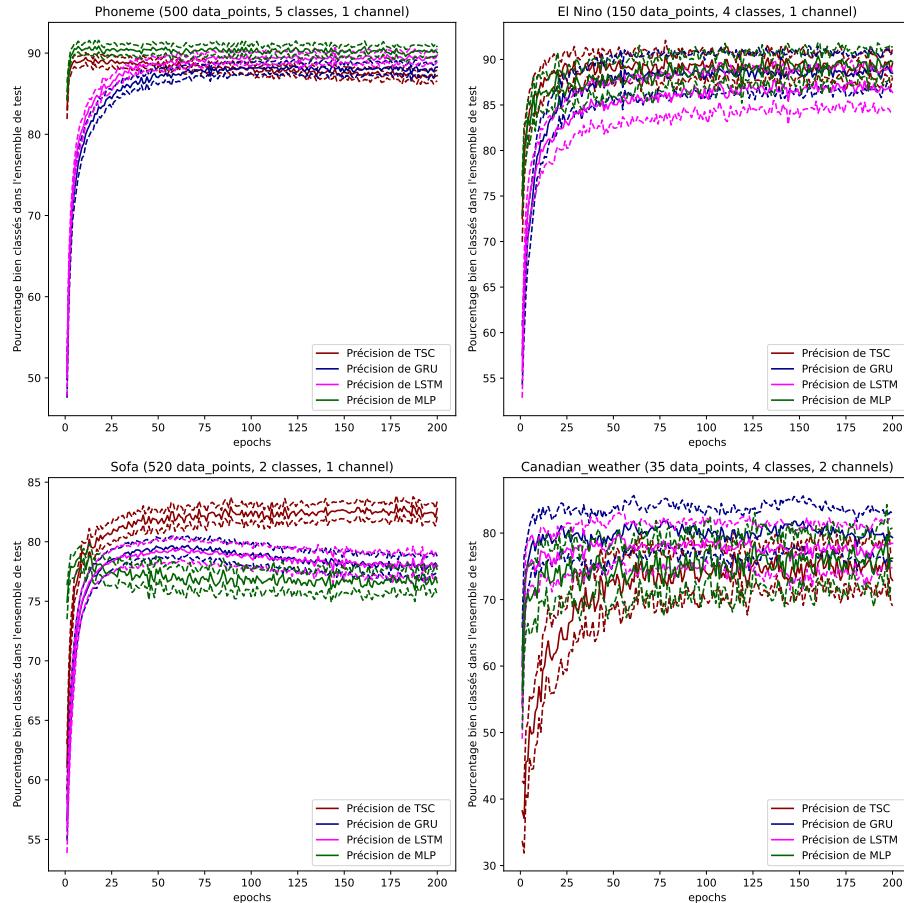


FIGURE 4.11 – Comparaison avec intervalles de confiance. MLP est significativement plus performant que les autres modèles pour **phoneme** tout au cours de l’entraînement. TSC est significativement plus performant sur **SOFA**, à partir de la 25 ème époque. Puisqu’il s’agit du problème des données irrégulièrement espacées, on peut considérer que la solution apportée est meilleure que l’utilisation d’autres modèles. Aucune autre conclusion ne peut être tirée de manière significative.

Étant donné le fait que le modèle semble apprendre en moins d’époques, nous nous sommes demandés si TSC avait besoin de moins de données d’entraînement que les autres modèles pour faire de bonne prédictions. Ainsi nous avons réduit le nombre de données d’entraînement et calculé la précision maximale des modèles sur 100 époques sur l’ensemble de test. Ces entraînements ont été réalisés de la même manière que la méthode de Monte-Carlo décrite précédemment mais on calcule cette fois-ci la précision maximale en fonction du nombre de données d’entraînement, et on fait la moyenne sur le nombre de simulations. Pour 50 simulations, aucune différence statistiquement significative n’a été détectée,

et ce sur beaucoup d'exemples et d'hyper-paramètres. La figure 4.12 présente un exemple des figures obtenues lors de la comparaison des modèles avec cette méthode.

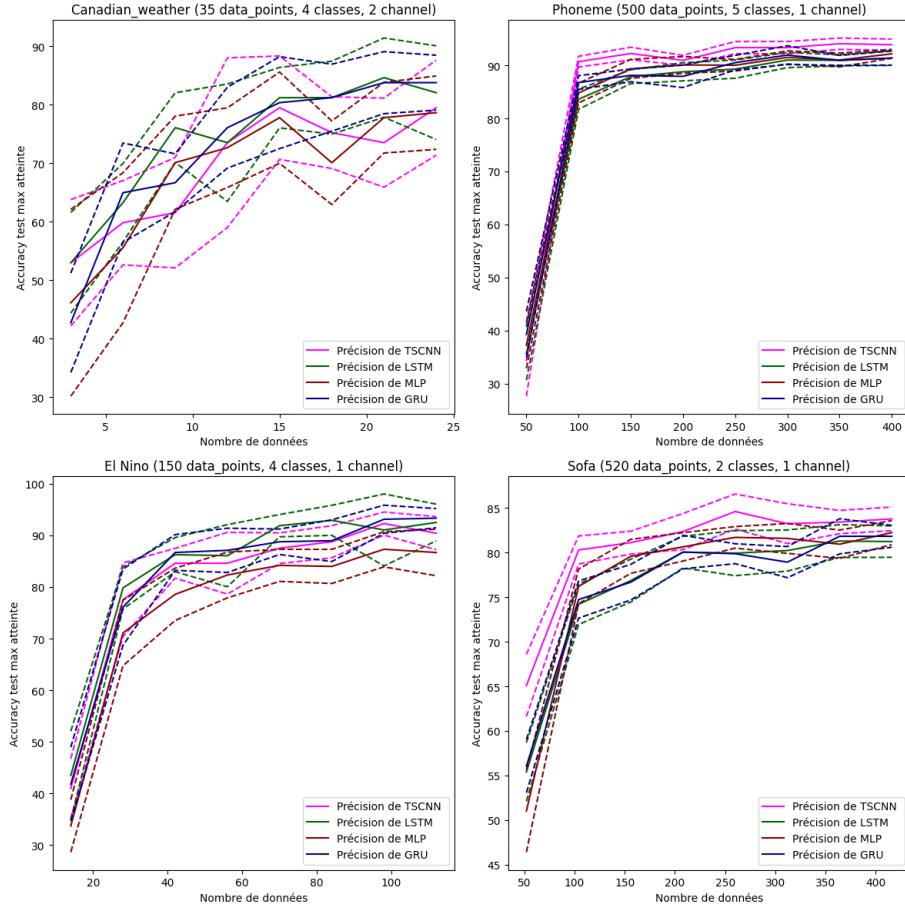


FIGURE 4.12 – Comparaison des modèles en fonction du nombre de données d'entraînement aucune différence significative n'est relevée. Le réseau de neurones récurrent Long-Short Term Memory est très légèrement meilleur sur les données El nino et sur Canadian Weather.

3.3 Un exemple de régression :

Nous présentons brièvement ici les meilleurs résultats obtenus pour la régression avec les meilleurs paramètres trouvé pour le modèle de convolution lissée ajusté, comparé aux autres modèles. On utilise le jeu de donnée Tecator qui est une régression sur trois variables de protéine, glucide et . La métrique choisie ici est la racine de l'écart quadratique à la réponse, *Root Mean Square Error* (RMSE), et malgré des efforts d'optimisation des hyper-paramètres, la puissance de calcul

dont nous disposons ne nous a pas permis d'avoir les même paramètres que sur les jeux de données précédents, et la limite de n pour un temps de calcul raisonnable et surcharger la mémoire est de l'ordre des centaines. Nous n'obtenons donc pas des performances satisfaisantes contrairement à la classification. Il est possible d'imaginer qu'avec une mémoire suffisante et plusieurs GPU, une meilleure performance est très certainement atteignable.

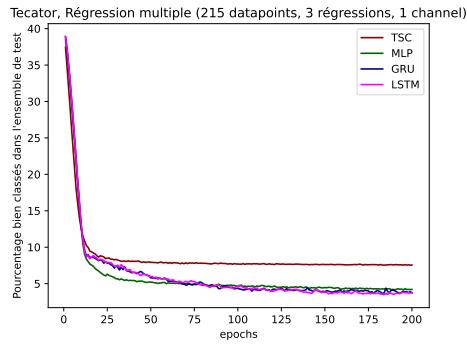


FIGURE 4.13 – Comparaison le sur jeu de données Tecator. Pour la régression aucun doute possible, TSC est le moins bon des modèles. De plus la durée de son entraînement sur ce jeu de données est 2 à 3 fois plus long que l'entraînement des autres modèles, ce qui n'en fait pas un bon choix pour ce jeu de données.

Conclusion :

Créer une couche d'entrée pour réseau de neurones qui puisse s'appliquer aux données fonctionnelles présente de nombreux défis de modélisation dont le plus grand d'entre eux est sans doute la dimension infinie supposée des variables à l'origine des observations, ce qui en pratique est difficilement représentable tel quel. Nous proposons une solution à ce problème en offrant un large choix de représentations, considérant cette dernière comme un hyper-paramètre supplémentaire aux réseaux de neurones convolutifs. L'avantage de la solution que nous proposons est de mettre à disposition un continuum entre la réduction de dimension maximale que peut être la convolution appliquée aux données brutes, et le filtrage fonctionnel continu, en évaluant les variables fonctionnelles à une précision arbitrairement grande. Notre contribution principale n'est pas dans l'amélioration des performances de l'état de l'art actuel en matière de réseaux de neurones pour l'analyse de données fonctionnelles, bien qu'elle soit un effet secondaire de la résolution du problème des données irrégulièrement espacées, mais bien dans la flexibilité paramétrique et les nombreuses options de modélisation qu'offre la convolution lissée ajustable. Elle offre également l'ajout des dérivées au modèle, ce qui n'est pas le cas dans les modèles actuels d'apprentissage profond en FDA. Elle est de

plus facile à mettre en place de part le fait qu'elle utilise des outils déjà implémentés en machine, et une implémentation simple est disponible à ce [lien github](#), ainsi que tous les codes ayant permis les expériences de la partie 4. Il est tout de même à noter que dans la pratique, notre modèle est beaucoup plus coûteux en calcul sur des jeux de données comportant peu de données et peu de mesures par donnée et donc nécessitant peu de paramètres à apprendre pour un MLP par exemple. Cependant le fait qu'il généralise également le réseau de neurone dense (en ne prenant qu'un seul filtre que l'on ne déplace pas et qui prend en entrée les données brutes sur un lissage par interpolation de manière similaire à la figure 3.2, avec un seul noyau de taille T_0) nous permet d'affirmer qu'il peut être au moins aussi performant que MLP si on le décide.

Le modèle que nous proposons a pour avantages certes de réduire le nombre de paramètres que la machine doit apprendre et d'être très maniable, cependant il a le désavantage d'être trop maniable, et il peut être intéressant à l'avenir d'étudier comment choisir les bases d'expansion pour augmenter les performances du modèles, ainsi que d'étudier les impacts effectifs de la grille d'évaluation choisie. En effet, même si les résultats théoriques du modèles sont prometteurs, sa performance n'est pas significativement meilleure que les modèles existant avec les paramètres que j'ai pu mettre en place à part sur les données **SOFA** qui sont celles qui représentent le mieux le défi des données irrégulièrement espacées. On peut également envisager au lieu d'ajouter des 0 pour MLP de considérer les temps non mesurés des observations comme des données manquantes et d'utiliser une méthode d'imputation comme missForest [11].

Notre objectif est également avec ce projet de convaincre des ingénieurs et autres scientifiques avec plus d'expérience dans le choix d'hyper-paramètres ou dans l'analyse de données fonctionnelles de s'intéresser à l'optimisation des modèles TSC qui offrent une grande flexibilité de choix de modélisation. Il peut être envisager de s'intéresser en détail au problème de la régression qui semble être un point faible de ces modèles, mais encore une fois, les hyper-paramètres étant tellement nombreux, un modèle efficace existe très certainement.

BIBLIOGRAPHIE

- [1] S Ben Driss, Mahmoud Soua, Rostom Kachouri, and Mohamed Akil. A comparison study between mlp and convolutional neural network models for character recognition. In *Real-Time Image and Video Processing 2017*, volume 10223, pages 32–42. SPIE, 2017.
- [2] Alessandro Giusti, Dan C Cireşan, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. In *2013 IEEE international conference on image processing*, pages 4034–4038. IEEE, 2013.
- [3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [4] Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE international conference on signal and image processing applications (ICSIPA)*, pages 342–347. IEEE, 2011.
- [5] Carlos Ramos-Carreño, José Luis Torrecilla, Miguel Carbajo-Berrocal, Pablo Marcos, and Alberto Suárez. scikit-fda : a python package for functional data analysis. *arXiv preprint arXiv :2211.02566*, 2022.
- [6] JO Ramsay and BW Silvermann. Functional data analysis. springer series in statistics, 1998.
- [7] Fabrice Rossi, Nicolas Delannay, Brieuc Conan-Guez, and Michel Verleysen. Representation of functional data in neural networks. *Neurocomputing*, 64 :183–210, 2005.
- [8] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization ? *Advances in neural information processing systems*, 31, 2018.

- [9] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning : From theory to algorithms*. Cambridge university press, 2014.
- [10] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D : Nonlinear Phenomena*, 404 :132306, 2020.
- [11] Daniel J Stekhoven and Peter Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1) :112–118, 2012.
- [12] Sachin S Talathi and Aniket Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv :1511.03771*, 2015.
- [13] Barinder Thind, Kevin Multani, and Jiguo Cao. Deep learning with functional inputs. *Journal of Computational and Graphical Statistics*, 32(1) :171–180, 2023.
- [14] Haixu Wang and Jiguo Cao. Functional nonlinear learning. *Journal of Computational and Graphical Statistics*, (just-accepted) :1–32, 2023.
- [15] Sidi Wu, Cédric Beaulac, and Jiguo Cao. Neural networks for scalar input and functional output. *Statistics and Computing*, (33), 2023.

Remerciements :

Je tiens à témoigner toute ma reconnaissance aux personnes qui m'ont permis de réaliser ce rapport de recherche ainsi que l'expérience que fut mon stage et voyage.

Tout d'abord un grand merci au professeur Cédric Beaulac qui est à l'origine des idées majeures de ce travail et qui m'a laissé une grande liberté de recherche et d'exploration basée sur des recommandations de littératures très pertinentes. Il a eu su faire naître en moi un goût spécifique pour la recherche et l'autonomie nécessaire dans cette discipline, tout en m'ayant donné la liberté d'explorer des idées et de collaborer avec d'autres étudiants et chercheurs.

Je tiens à également témoigner ma gratitude à **Marilou Brickert**, étudiante en design graphique à l'université Concordia, qui est à l'origine de toutes les animations visuelles disponibles dans ce rapport, sans lesquels la pédagogie de ce dernier aurait été bien moindre.

Je remercie enfin mes parents qui m'ont soutenu dans le choix de projet malgré les nombreuses barrières qui se sont dressées devant moi quand à la mobilité pour un stage de fin de second cycle, et sans lequel ce projet n'aurait pas pu voir le jour.

CHAPITRE ANNEXE

Les [ressources](#) en terme de jeux de données d'analyse de données fonctionnelles sont disponibles pour la plupart sur le [CRAN](#) ou directement dans la bibliothèque *Python scikit-fda*.

Les codes utilisés pour développer le modèles et le tester sont disponibles sur mon [github](#), la comparaison des modèles nécessitant un version récente de CUDA et donc une carte graphique compatible. Une version plus simple du modèle peut-être testée dans le [notebook ci-joint](#).