Approximation de Dynamic Time Warping par réseaux de neurones pour la compression de signaux EEG et l'analyse de l'insomnie induite par le COVID long.

Hugo Lerogeron

Normandie Université

UNIVERSITÉ DE ROUEN NORMANDIE

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité **INFORMATIQUE**

Préparée au sein de l'**Université de Rouen Normandie**

Approximation de Dynamic Time Warping par réseaux de neurones pour la compression de signaux EEG et l'analyse de l'insomnie induite par le COVID long.

Présentée et soutenue par

HUGO LEROGERON

Thèse soutenue le 21/12/2023

devant le jury composé de :

M. LAURENT OUDRE	Professeur des Universités - Ecole Normale Supérieure Paris-Saclay	Rapporteur du jury
M. ROMAIN TAVENARD	Professeur des Universités - UNIVERSITE DE HAUTE BRETAGNE RENNES 2	Rapporteur du jury
M. CÉCILE CAPPONI	Professeur des Universités - Aix-Marseille université	Membre du jury
M. ROMAIN PICOT-CLEMENTE	-	Membre du jury
M. ALAIN RAKOTOMAMONJY	Professeur des Universités - Université de Rouen Normandie	Membre du jury
MME LATIFA OUKHELLOU	Directeur de Recherche - UNIVERSITE MARNE LA VALLEE UNIVERSITE MARNE LA VALLEE	Président du jury
M. LAURENT HEUTTE	Professeur des Universités - Université de Rouen Normandie	Directeur de thèse

Thèse dirigée par **LAURENT HEUTTE** (LABORATOIRE D'INFORMATIQUE DE TRAITEMENT DE L'INFORMATION ET DES SYSTEMES)

MIIS

litis

Remerciements

Premièrement, je remercie mon directeur de thèse Laurent Heutte de s'être intéressé à mes travaux durant l'intégralité de la thèse. Son exigence, mais aussi ses encouragements et conseils, m'ont permis de m'améliorer durant ces trois ans, notamment concernant les manières de bien communiquer et de mettre en avant mes travaux. Je remercie également mon encadrant dans l'entreprise Saagie, Romain Picot-Clémente, d'avoir cru en moi et facilité le lancement de ma thèse, mais aussi de son travail continu pour me guider durant mes recherches. Je remercie aussi Alain Rakotomamonjy pour ses conseils concernant mes travaux durant l'intégralité de ma thèse. J'adresse de plus mes remerciements à Latifa et Samia, qui au travers de leur participation à mon CSI m'ont permis de mener à bien mes travaux.

Je tiens à remercier mes collègues de Saagie. En particulier Miriam, qui du fait de son expérience de doctorante CIFRE m'a beaucoup aidé, entre autres avec les aspects administratifs de la thèse. Je remercie Sébastien, ancien membre du pôle recherche, qui m'a permis d'intégrer l'entreprise, ainsi que les membres actuels, Baptiste et LeShan. Leur sympathie et leur curiosité ont permis de rendre les journées de travail plus productives et agréables. Je remercie de plus les fans d'escalade : Antoine, Tony, Alan et Guillaume des fantastiques moments passés à se blesser les doigts : ce sport a grandement participé à maintenir ma motivation durant la thèse.

Je remercie mes amis, Mathilde, Margaux, Léa, Julie, Antoine, Tristan et Maxime pour leur positivité et leurs encouragements durant ma thèse, en particulier dans les moments difficiles.

Enfin, je remercie Margot pour son aide et son soutien indéfectible avant, pendant et après la thèse.

Abstract

This manuscript presents the work carried out within the framework of the CIFRE thesis conducted in partnership between LITIS and Saagie and which is part of the PANDORE-IA project in association with the VIFASOM sleep center.

Electroencephalographic (EEG) signals are very useful in helping experts identify various abnormalities like sleep disorders. Recently, the community has shown great interest in long COVID and its various impacts on sleep. However, these signals are voluminous : compression allows reducing storage and transfer costs. Recent compression approaches are based on autoencoders that use a cost function to learn. It is usually the Mean Squared Error (MSE), but there are metrics more suited to time series, particularly Dynamic Time Warping (DTW). However, DTW is not differentiable and thus can not be used as a loss for end-to-end learning.

To solve this problem, we propose in this thesis two approaches to approximate DTW based on neural networks. The first approach uses a Siamese network to project the signals so that the Euclidean distance of the projected signals is as close as possible to the DTW of the original signals. The second approach attempts to predict directly the DTW value. We show that these approaches are faster than other differentiable approximations of DTW while obtaining results similar to DTW in query or classification on sleep data.

We then demonstrate that the Siamese approximation can be used as a cost function for learning a sleep signal compression system based on an autoencoder. We justify the choice of the network architecture by the fact that it allows us to vary the compression rate. We evaluate this compression system by classification on the compressed and then reconstructed signals, and show that the usual measures of compression quality do not allow for a proper assessment of a compression system's ability to retain discriminative information. We show that our DTW approximations yield better performance on the reconstructed data than conventional compression algorithms and other reconstruction losses.

Finally, to study the impact of long COVID on insomnia, we collect and provide the community with a dataset named COVISLEEP, containing polysomnographies of individuals who developed chronic insomnia after COVID infection, and of those suffering from chronic insomnia but who have not been infected by the virus. We compare various state-of-the-art approaches for sleep staging, and use the best one

for learning the detection of long COVID. We highlight the difficulty of the task, especially due to the high variability among patients. This offers a complex dataset to the community that allows for the development of more effective methods.

Résumé

Ce manuscrit présente les travaux effectués dans le cadre d'une thèse CIFRE réalisée en collaboration entre le LITIS et Saagie, et qui s'inscrit dans le projet PANDORE-IA en partenariat avec le centre du sommeil VIFASOM.

Les signaux électroencéphalographiques (EEG) sont très utiles pour aider les experts à identifier diverses anomalies comme les troubles du sommeil. En particulier dernièrement, la communauté s'est beaucoup intéressée au COVID long et à ses divers impacts sur le sommeil. Ces signaux sont cependant volumineux : la compression permet de réduire les coûts de stockage et de transfert. Les approches récentes de compression se basent sur des autoencodeurs qui utilisent une fonction de coût pour apprendre. Celle-ci est usuellement la MSE, mais il existe des métriques plus adaptées aux séries temporelles, en particulier DTW. DTW n'est toutefois pas différentiable et ne peut donc être utilisée pour un apprentissage de bout-en-bout.

Pour résoudre ce problème, nous proposons dans cette thèse deux approches d'approximation de DTW basées sur des réseaux de neurones. La première approche utilise un réseau siamois pour projeter les signaux de sorte que la distance euclidienne des signaux projetés soit la plus proche possible de la DTW des signaux originaux. La deuxième approche tente de directement prédire la valeur de DTW. Nous montrons que ces approches sont plus rapides que les autres approximations différentiables de DTW tout en obtenant des résultats similaires à l'utilisation de DTW dans des tâches de requêtage ou de classification de signaux du sommeil.

Nous montrons ensuite que l'approximation siamoise peut être utilisée comme fonction de coût pour apprendre un système de compression des signaux de sommeil basé sur un autoencodeur. Nous justifions du choix de l'architecture du réseau par le fait qu'elle nous permet de faire varier le taux de compression. Nous évaluons ce système de compression par la classification sur les signaux compressés puis reconstruits, et montrons que les mesures usuelles de qualité de compression ne permettent pas de correctement évaluer la capacité d'un système de compression à conserver l'information discriminante. Nous montrons que nos approximations de DTW permettent d'obtenir de meilleures performances sur les données reconstruites que des algorithmes de compression usuels et que d'autres fonctions de coût de reconstruction.

Enfin, pour étudier l'impact du COVID long sur l'insomnie, nous collectons et mettons à disposition de la communauté un jeu de données nommé COVISLEEP, constitué

de polysomnographies de personnes ayant développé une insomnie chronique après infection du COVID, et de personnes souffrant d'insomnie chronique mais n'ayant pas été infectées par le virus. Nous comparons diverses approches à l'état de l'art pour classifier les états du sommeil, et utilisons la meilleure pour apprendre la détection de COVID long. Nous montrons la difficulté de la tâche, notamment due à la forte variabilité entre les patients. Ceci offre à la communauté un jeu complexe qui laisse place au développement de méthodes plus performantes.

Table des matières

Remerciements **1**

Abstract **3**

Résumé **5**

Liste des tableaux **12**

Liste des figures **15**

1 Introduction **17**
1.1 Contexte de l'étude et motivations 17
1.2 Contributions 19
1.2.1 Approximation de DTW 19
1.2.2 Compression de signaux de sommeil 19
1.2.3 Détection de l'impact du COVID long sur l'insomnie 20
1.3 Organisation du manuscrit 20

2 Classification de séries temporelles : contexte et cas particulier du sommeil **23**
2.1 Classification de séries temporelles 25
2.1.1 Méthodes par modèles 25
2.1.2 Méthodes par distances 26
2.1.2.1 Distance euclidienne 26
2.1.2.2 Dynamic Time Warping 26
2.1.2.3 Autres distances 27
2.1.2.4 Utiliser la distance pour classifier 29
2.1.3 Méthodes par création de caractéristiques 30
2.1.3.1 Hand-crafted features 30
2.1.3.2 Méthodes par apprentissage profond 31
2.1.3.3 Types de réseaux pour les séries temporelles 36
2.1.3.4 Le manque de données labellisées 43
2.1.4 Conclusion 48
2.2 Classification de sommeil 49
2.2.1 Principaux jeux de données 49
2.2.1.1 SHHS 49

2.2.1.2 SleepEDF . . . 50
2.2.1.3 Autres jeux de données . . . 51
2.2.2 Métriques . . . 52
2.2.2.1 Accuracy . . . 52
2.2.2.2 Macro F1 . . . 52
2.2.2.3 Kappa de Cohen . . . 53
2.2.3 Architectures de modèles et comparaisons . . . 53
2.3 Conclusion et discussion . . . 59

3 Approximation de DTW via un réseau de neurones convolutionnel 61
3.1 Introduction . . . 63
3.1.1 Contexte et motivations . . . 63
3.1.2 Définition du problème . . . 64
3.2 État de l'art . . . 65
3.2.1 Approximer pour accélérer DTW . . . 65
3.2.1.1 FastDTW . . . 66
3.2.1.2 ucrDTW . . . 67
3.2.2 Approximer pour rendre DTW différentiable . . . 69
3.2.2.1 Non-différentiabilité de DTW . . . 70
3.2.2.2 softDTW . . . 70
3.2.2.3 DTWNet . . . 72
3.2.3 Approximation de fonction via réseaux de neurones . . . 73
3.2.4 Approximation de la distance de Wasserstein via un réseau siamois 74
3.2.5 LDPS . . . 75
3.2.6 Conclusion . . . 77
3.3 Notre méthode pour approximer DTW . . . 77
3.3.1 Architecture siamoise . . . 77
3.3.2 Procédure d'apprentissage . . . 79
3.3.3 Choix de l'encodeur . . . 80
3.3.3.1 SorsNet . . . 80
3.3.4 Choix du décodeur . . . 81
3.3.5 Architecture directe . . . 81
3.3.6 Conclusion . . . 82
3.4 Expérimentations . . . 82
3.4.1 Jeu de données . . . 83
3.4.1.1 Prétraitements . . . 83
3.4.1.2 Matrice de vérité terrain . . . 83
3.4.2 Implémentations et apprentissage . . . 84
3.4.3 Vitesse . . . 85
3.4.4 Fidélité . . . 87
3.4.4.1 Plus proche voisin . . . 87
3.4.4.2 Classification via k plus proche voisin (KPPV) . . . 89
3.4.5 Différentiabilité . . . 90
3.4.6 Conclusion . . . 91
3.5 Application à d'autres jeux de données . . . 91
3.5.1 Recherche de séries similaires . . . 92
3.5.2 Classification 1-PPV . . . 93

3.5.3 Classification de sommeil via prototypes . . . 93
3.6 Synthèse des résultats . . . 93
3.7 Conclusion du chapitre . . . 96

4 Compression de signaux de sommeil via approximation de DTW 97
4.1 Introduction . . . 98
4.1.1 Contexte et motivations . . . 98
4.1.2 Définition du problème . . . 99
4.1.3 Métriques . . . 100
4.2 État de l'art . . . 101
4.2.1 Approches basées sur des dictionnaires . . . 102
4.2.2 Approches basées sur la transformation par fonction . . . 104
4.2.3 Approches par autoencodeurs . . . 106
4.2.3.1 Architecture de l'encodeur et du décodeur . . . 106
4.2.3.2 Le choix de la fonction de coût de reconstruction . . . 109
4.2.4 Conclusion . . . 109
4.3 Méthode proposée pour compresser les signaux . . . 110
4.3.1 Architecture générale . . . 110
4.3.2 Choix de la fonction de coût . . . 112
4.3.3 Architecture de l'encodeur-décodeur . . . 112
4.3.4 Apprentissage . . . 113
4.4 Résultats expérimentaux . . . 113
4.4.1 Comparaisons avec les approches non paramétriques . . . 113
4.4.2 Classification sur les données reconstruites . . . 114
4.4.2.1 Apprentissage de la classification de sommeil . . . 115
4.4.2.2 Sur SleepEDF-78 . . . 116
4.4.2.3 Avec d'autres classifieurs . . . 117
4.4.2.4 Sur un autre jeu de données . . . 118
4.5 Conclusion . . . 119

5 Projet PANDORE-IA : détection de l'impact du COVID long sur le sommeil 121
5.1 Introduction . . . 122
5.2 Projet PANDORE-IA . . . 123
5.2.1 Acteurs du projet . . . 123
5.2.2 Objectifs scientifiques du projet . . . 124
5.3 Etat de l'art : détection du COVID long . . . 125
5.4 Jeu de données COVISLEEP . . . 126
5.4.1 Acquisition du jeu de données . . . 126
5.4.2 Statistiques . . . 128
5.4.3 Préparation du jeu de données . . . 128
5.5 Classification du sommeil . . . 129
5.5.1 Modèles comparés . . . 130
5.5.2 Résultats . . . 134
5.6 Identification du COVID long . . . 137
5.6.1 Statistiques de sommeil . . . 137
5.6.2 Classification . . . 137

5.6.2.1 L'impact du COVID long est-il plus prévalent dans un stade de sommeil spécifique ? 142
5.6.2.2 Variabilité inter-patients 142
5.7 Conclusion du chapitre 143

6 Conclusion et Perspectives **145**
6.1 Perspectives 147
6.1.1 Approches multidimensionnelles et généralisation 147
6.1.2 Explicabilité de la détection du COVID long 147

Liste des communications **149**

Bibliographie **149**

Liste des tableaux

2.1 Répartitions des labels selon les classes et nombre total de labels des jeux de données SHHS-1 et SC-EDF-78. 51

2.2 Pourcentage de transition entre les différents stades du sommeil du jeu de données SleepEDF-78 54

2.3 Score macro F1 de différents modèles de classification de sommeil sur différents jeux de données. Extrait et adapté de [126], où nous avons remplacé le score κ par le score MF1. 58

3.1 Pourcentage de fois que le signal le plus proche d'un signal donné q selon une métrique est dans le top 5 des signaux les plus proches de q selon DTW. N_t est le nombre de signaux dans le jeu de test. 87

3.2 Pourcentage de fois que le signal le plus proche d'un signal donné q selon une métrique est dans le top 5 des signaux les plus proches de q selon DTW. N_t est le nombre de signaux dans le jeu de test. Les signaux sont ré-échantillonnés pour être de taille $L \in [500, 1000]$. 88

3.3 Score Macro F1 et *accuracy* de la classification d'état du sommeil sur SleepEDF-78 par 1-PPV en utilisant différentes métriques. On moyenne le résultat de 3 itérations. Pour chaque itération donnée, toutes les méthodes utilisent les mêmes sous-parties du jeu de données. 89

3.4 Score de recherche de plus proche voisin sur SHHS, comparant notre architecture directe apprise sur SHHS avec celle apprise sur SleepEDF. 92

3.5 Macro F1 et *accuracy* de classification de sommeil sur SHHS en ayant appris sur SleepEDF-78. Les classifieurs sont des 1-PPV basés sur les différentes métriques. Chaque résultat affiché est la moyenne de 3 itérations, avec l'écart type associé. 93

3.6 Synthèse des résultats des comparaisons d'approximation de DTW développées dans ce chapitre. Les valeurs de vitesse et de fidélité sont les rangs des méthodes. Les résultats de fidélité de LDPS et DeepDTW Siamese étant très similaires, les méthodes sont classées troisièmes égalité (3*). 95

4.1 Performances de différentes architectures de compression sur le jeu de données MIT-BIT. [42] utilisent des fenêtres plus longues ($N = 1000$). 108

4.2 Résultats d'algorithmes non paramétriques et de notre méthode selon les métriques usuelles de compression, sur le jeu SleepEDF-78 114

4.3 Score de classification d'état du sommeil sur SleepEDF-78. Les signaux sont compressés par différents modèles avec le facteur de compression CF puis décompressés avant d'être utilisées par un classifieur pour l'apprentissage et le test. La classification est apprise 3 fois et le score moyen est affiché. 117
4.4 Score de classification d'état du sommeil sur SleepEDF-78 par AttentionSleep sur signaux compressés puis reconstruits. 118
4.5 Score de classification d'état du sommeil sur SleepEDF-78 par 1D-CNN sur signaux compressés puis reconstruits. 118
4.6 Score de classification d'état du sommeil (5 classes) sur SHHS par SorsNet sur signaux compressés puis reconstruits avec un facteur de compression de 32. Nos méthodes et LDPS sont apprises sur SleepEDF-78. 119

5.1 Statistiques du jeu de données COVISLEEP. *Time in Bed* représente la durée totale de l'expérimentation, *Sleep Period Time* représente la durée entre la première et la dernière phase de sommeil de la nuit. *Sleep Time* est le temps réellement passé à dormir (états N1, N2, N3 et REM), la *Sleep Efficiency* est le ratio entre la durée de l'expérience et *Sleep Time*, *Sleep Onset Latency* indique la durée avant le premier endormissement, *Wake After Sleep Onset* indique la durée d'éveil après le premier endormissement. 128
5.2 Distribution de trios de patients dans les *folds*. 129
5.3 Macro F1/Accuracy de divers modèles en classification du sommeil sur le jeu COVISLEEP. 136
5.4 Accuracy en classification de signaux de sommeil, entre les signaux issus de personnes souffrant de COVID long (sujets) et les autres (contrôles). 139
5.5 Accuracy de la classification de signaux de sommeil, entre les signaux issus de personnes souffrant de COVID long et les autres. Les signaux sont sélectionnés selon l'état de sommeil associé. On présente les résultats pour tous les patients, puis uniquement les sujets, puis uniquement les contrôles. 141

Table des figures

2.1 Alignement de distance euclidienne et DTW. Par [168]. 27
2.2 Création de la matrice de caractéristiques via distance entre les séries temporelles. Extrait de [1]. Ici, les séries temporelles $x^{(i)}$ sont nommées TS_i, et les classes correspondantes C_i. 30
2.3 Early stopping basé sur l'erreur sur le jeu de validation. Extrait de [61]. 34
2.4 Exemple de perceptron.[1] 35
2.5 Multi Layer Perceptron. Extrait de[2] 35
2.6 RNN, compressé et déplié. Par [39]. 37
2.7 Cellule standard LSTM. Par [175]. 38
2.8 Convolution dilatée avec q=[1,2,4] et k = 3. Extrait de [11] 40
2.9 Skip connection. Extrait de [11], k est la taille du filtre de convolution, et q le facteur de dilatation. 40
2.10 Illustration du mécanisme d'attention sur séries temporelles. Par [95]. . 41
2.11 Architecture du *Transformer*. Par [176]. 42
2.12 Effet de certaines augmentations sur un signal EEG 44
2.13 Représentation d'un autoencodeur.[3] 45
2.14 Architecture de l'encodeur du TCN-AE. Extrait de [170]. Dans chaque bloc TCN, dConv indique une couche de convolution dilatée de taux q et de taille de noyau 64. Une couche de convolution 1×1 réduit ensuite la dimension des caractéristiques à 16. Les différentes représentations obtenues par les blocs TCN sont concaténées dans la partie concat1. . 46
2.15 Architecture du décodeur TCN-AE. La couche *upsamp* ré-échantillonne l'entrée du décodeur de sorte que le signal retrouve sa taille initiale (avant d'être encodé). Ensuite le fonctionnement est analogue à l'encodeur, avec différentes convolutions dilatées *dconv* puis une concaténation des représentations (*concat3*). 47
2.16 Architecture du DeepSleepNet [165]. Fs est la fréquence d'échantillonnage du signal d'entrée, et indique la taille de filtre des couches de convolutions (*conv*) initiales. *max-pool* indique des couches de Max-Pooling, et *bidirect-lstm* des couches de LSTM bidirectionnelles, avec une dimension de 512 pour l'espace latent. Les couches *dropout* sont précédées de leur probabilité. 55
2.17 Architecture du RobustSleepNet [66] 56
2.18 Architecture du SleepTransformer [131] 57

3.1 Restrictions de l'espace de recherche du chemin d'alignement dans la matrice de coût [4] 66
3.2 Borne inférieure de Keogh. En haut sont les séries à comparer, au milieu le *Wegde* obtenu, et en bas la borne de Keogh. Extrait de [84]. 68
3.3 Résultat de l'opérateur $softmin$ pour différentes valeurs de γ, en comparaison avec l'opérateur min. Inspiré par [168]. 71
3.4 Exemple de paradigme d'approximation d'équation différentielle partielle de Kolmogorov. Extrait de [20]. Les variables d'entrée x, t et γ sont échantillonnées de $X \sim \mathcal{D} \times [\nu, w]^d \times [0, T]$, où $\mathcal{D} \times [\nu, w]^d \times [0, T]$ est le domaine spatio-temporel d'intérêt de l'équation. On nomme les conditions initiales de l'équation φ_γ . La valeur cible $y = \varphi_\gamma(S_{\gamma,x,t}) \in \mathbb{R}$ est obtenue explicitement ou via des solveurs numériques (S) selon les types d'équations à résoudre. 73
3.5 Approximation de la distance de Wasserstein par [40] 75
3.6 Architecture globale de notre méthode. Deux signaux sont extraits du jeu de données et encodés. L'objectif est que la distance euclidienne entre les signaux encodés soit aussi proche que possible de la DTW entre les signaux originaux. Le décodeur tente ensuite de retrouver les signaux originaux à partir des représentations. 78
3.7 Architecture du SorsNet [159]. 81
3.8 Architecture globale de notre approximation directe. Deux signaux sont extraits du jeu de données puis concaténés pour obtenir l'entrée $X_c \in \mathbb{R}^{N,2}$. L'encodeur prédit ensuite directement la valeur de DTW, qui est comparée à la réalité par MSE. 82
3.9 Temps moyen nécessaire à calculer la métrique entre deux signaux de sommeil, en fonction de la longueur des signaux. Tous les calculs sont exécutés en Python sur CPU (Intel Xeon Silver 4112 CPU @ 2.60GHz). 86
3.10 *Accuracy* de la classification par prototype sur le jeu de validation pendant l'apprentissage des prototypes, en fonction de la métrique utilisée pour comparer les signaux et les prototypes. DTW et FastDTW sont utilisés pour obtenir le chemin nécessaire à la méthode de Cai et al [29]. 91
3.11 *Accuracy* de la classification par prototype sur le jeu de validation de SHHS pendant l'apprentissage des prototypes, en fonction de la métrique utilisée pour comparer les signaux et les prototypes. 94

4.1 Schéma du *pipeline* de classification de données compressées puis reconstruites. 111
4.2 Zoom sur l'architecture du système de compression. 111
4.3 Comparaison entre les signaux originaux et reconstruits après compression ($CF = 32$) et décompression par notre méthode (TCN-AE avec DeepDTW Siamese comme fonction de coût), DCT, FFT-r et DWT . . 115

5.1 Extraction de caractéristiques selon [174]. $x^{(t)}$ est la fenêtre centrale. On ajoute les caractéristiques calculées sur les 2 fenêtres précédentes et suivantes, ainsi que sur la concaténation des fenêtres adjacentes. 131
5.2 Architecture globale du modèle AttentionSleep. Extrait de [54] 131

5.3 Architecture de l'extracteur de caractéristiques d'AttentionSleep. Extrait de [54]. Conv1D (64,50,6) désigne une couche de convolution 1D avec 64 filtres, une taille de noyau de 50 et un *stride* de 6. GELU est une fonction d'activation introduite par [70]. 132
5.4 Rappel de l'architecture du DeepSleepNet [165]. Le modèle est décrit dans le détail dans la section 2.2.3. 133
5.5 Architecture de SleepyCo. Extrait de [93] 134
5.6 Statistiques de sommeil des patients sujets et contrôles 138
5.7 Accuracy de la classification de signaux de sommeil par la méthode Sleep-Linear avec CatBoost, entre les signaux issus de personnes souffrant de COVID long (sujets) et les autres (contrôles), pour chaque patient. La ligne pointillée représente le seuil de 50% de précision (c'est-à-dire que 50% des fenêtres d'un individu sont bien classifiées). 140

Chapitre 1

Introduction

1.1 Contexte de l'étude et motivations

La compréhension du fonctionnement du cerveau est une quête ancienne et continue de l'humanité. Les neurosciences ont permis d'énormes progrès dans la compréhension de la structure et du fonctionnement de cet organe complexe. Cependant, il reste encore beaucoup à découvrir, notamment en ce qui concerne la manière dont le cerveau traite l'information. Les signaux électroencéphalographiques (EEG) sont un outil précieux pour explorer cette question. L'EEG permet d'enregistrer l'activité électrique du cerveau à travers le cuir chevelu, offrant ainsi une fenêtre sur les processus neuronaux en temps réel. Les signaux EEG sont généralement enregistrés sur une période et consistent en une série de mesures de tension. Largement utilisés dans des domaines tels que la neuroscience, la psychologie ou l'analyse du sommeil, l'étude de ces signaux peut fournir des informations précieuses sur la fonction cérébrale et aider les experts à identifier diverses anomalies comme des troubles du sommeil ou la maladie d'Alzheimer. En particulier dernièrement, la communauté s'est beaucoup intéressée au COVID et à ses divers impacts sur la santé. Dans le cas du COVID long, c'est-à-dire des symptômes du COVID persistants au-delà de 4 semaines [1], certains patients se plaignent de dégradation de la qualité de leur sommeil. Ainsi, cette thèse CIFRE, collaboration entre Saagie et le LITIS, s'inscrit dans le projet PANDORE-IA (financé dans le contexte d'un projet RAPID par la DGA) en partenariat avec VIFASOM. VIFASOM [2] est une unité de recherche située à l'Hôpital Hôtel-Dieu, sous tutelle de l'Université Paris-Cité et de l'Institut de Recherche Biomédicale des Armées. L'un des objectifs du projet PANDORE-IA est ainsi de comprendre l'impact du COVID long sur les EEG de sommeil de patients souffrant d'insomnies.

Cependant, l'analyse des données EEG n'est pas une tâche facile. Les signaux sont souvent bruités et contiennent une grande variabilité due aux différences d'activité cérébrale entre les individus. Cette variabilité rend difficile la comparaison et l'analyse

1. https://www.has-sante.fr/jcms/p_3237041/fr/symptomes-prolonges-suite-a-une-covid-19-de-l-adulte-diagnostic-et-prise-en-charge
2. https://www.frcneurodon.org/unite-vifasom/

des signaux EEG. Dans le cas du projet PANDORE-IA, les données sont de plus volumineuses, rendant leur traitement lent et coûteux. Ainsi, compresser les données pour réduire leur taille en perdant le moins d'information possible est une solution envisagée dans le cadre du projet. Cette idée sera explorée dans cette thèse.

L'extrême développement des réseaux de neurones ces dernières années a significativement impacté l'analyse de signaux temporels dans tous les domaines d'application. Ces réseaux apprennent à représenter les signaux dans un espace latent pour mieux réaliser la tâche voulue, qui peut être la classification, la détection d'anomalie ou la compression, entre autres. Pour améliorer cette représentation, il est nécessaire d'exploiter la propriété temporelle des signaux. Ainsi, les mécanismes de récurrence et d'attention sont communs dans l'architecture de réseaux analysant des signaux temporels. Un autre élément important pour l'apprentissage de cette représentation est la fonction de coût, qui mesure l'écart entre les prédictions faites par le modèle et la vérité attendue, ce qui donne la direction d'évolution des poids du modèle à apprendre. En particulier, le principe de la compression dans le cas de réseaux de neurones est souvent de compresser puis de restaurer un signal et d'essayer de minimiser les erreurs commises sur le signal restauré par rapport au signal initial. De fait, dans les tâches nécessitant de comparer la sortie du réseau à son entrée, il semble naturel de supposer que prendre en compte le caractère temporel des signaux permet de mieux évaluer le modèle et donc d'améliorer ses performances. Nous souhaitons donc utiliser une métrique adaptée aux signaux temporels.

L'algorithme Dynamic Time Warping [146] (DTW) permet de comparer deux séries temporelles, même si elles sont de longueurs différentes ou décalées temporellement. DTW cherche l'alignement optimal entre deux signaux, ce qui implique de les étirer ou de les comprimer pour les faire correspondre. L'algorithme est utilisé dans de nombreuses applications utilisant divers types de séries temporelles, à l'image par exemple de [154] qui utilise DTW pour reconnaître des activités humaines, ou de [118] qui utilise DTW pour vérifier l'authenticité de signatures. Cependant, DTW est non différentiable et donc inutilisable comme fonction de coût pour un apprentissage de bout en bout d'un modèle de réseau de neurones. L'algorithme est de plus de complexité quadratique par rapport à la taille des séries, rendant les calculs lents pour des séries de longue taille comme les EEGs.

Pour contourner ces difficultés, l'approche habituelle est d'approximer l'algorithme DTW. Une approximation est classiquement évaluée selon deux critères : la fidélité des résultats obtenus à ceux de l'algorithme original et le gain de temps permis par l'approximation. À cela s'ajoute dans le contexte de l'apprentissage profond la différentiabilité de l'approximation. Ainsi, beaucoup d'approximations de DTW existent dans la littérature, certaines ayant pour objectif d'accélérer l'algorithme, comme FastDTW [150], d'autres de le rendre différentiable, comme softDTW [41], en restant le plus fidèle possible. Satisfaire les trois critères conjointement est difficile : modifier l'algorithme pour l'accélérer nuit à sa fidélité, et contourner les points de non-différentiabilité ne permet pas à priori de rendre l'algorithme plus rapide. Enfin, mettre bout à bout diverses modifications risque de rendre le résultat de moins en moins fiable. De fait, peu d'approximations essaient d'être performantes sur les trois critères, fidélité, différen-

tiabilité et vitesse, en même temps.

Dans la plupart des cas où la méthode Dynamic Time Warping (DTW) est employée (par exemple la recherche de séries similaires, ou la classification de séries temporelles), l'objectif principal n'est pas d'utiliser l'alignement spécifique des signaux que DTW génère. En d'autres termes, ce qui importe le plus, c'est la valeur numérique résultante de DTW, qui sert de mesure de similarité entre deux séquences ou signaux. Ainsi, pour développer une méthode d'approximation efficace de DTW, il n'est pas nécessaire de reproduire fidèlement tout le processus de l'algorithme original. On peut simplement considérer DTW comme une « boîte noire »qui prend deux signaux en entrée et donne une valeur en sortie. Cette valeur sera alors la cible de notre méthode d'approximation.

Via ce paradigme, nous proposons d'approximer le résultat de DTW via un réseau de neurones, théoriquement capable d'approximer n'importe quelle fonction. Une fois cette approximation obtenue et validée, nous pourrons l'utiliser comme fonction de coût pour apprendre un système de compression.

1.2 Contributions

Cette thèse est principalement centrée sur l'étude de signaux de sommeil, pour la classification d'état du sommeil ainsi que la compréhension de l'impact du COVID long sur l'insomnie. Les enregistrements de sommeil étant de longues séries temporelles, nous cherchons d'abord à compresser les séries en perdant le minimum d'informations possible. Pour ce faire, nous proposons d'approximer DTW via un réseau de neurones, puis de s'en servir comme fonction de coût de reconstruction pour la compression.

1.2.1 Approximation de DTW

Nous proposons d'approximer DTW via un autoencodeur basé sur un réseau siamois. L'encodeur projette les signaux de telle sorte que la distance euclidienne entre les signaux projetés soit proche de la DTW entre les signaux originaux. Nous montrons qu'une telle approximation est fidèle à l'algorithme original, différentiable et peu impactée en temps de calcul par la longueur des séries. Nous comparons deux architectures, directe et siamoise, pour l'autoencodeur et explorons les avantages et inconvénients de chacune. Enfin, nous apprenons notre modèle d'approximation sur des données de sommeil EEGs et montrons qu'un tel modèle obtient d'excellentes performances sur des jeux de données similaires sans *finetuning*.

1.2.2 Compression de signaux de sommeil

Nous montrons ensuite qu'une telle approximation de DTW peut être utilisée comme fonction de coût de reconstruction pour apprendre la compression de signaux de sommeil, en apprenant un système de compression de bout en bout via un autoencodeur dont la fonction de coût est notre méthode d'approximation de DTW. En choisissant une architecture basée sur des couches de convolutions, nous pouvons de plus faire varier le taux de compression de notre modèle. Nous montrons qu'une telle méthode

permet de mieux classifier les états de sommeil des signaux reconstruits par rapport à d'autres fonctions de coût de reconstruction, avec plusieurs jeux de données et différents classifieurs. Nous montrons que les métriques usuelles pour évaluer la qualité de la compression ne sont pas suffisantes et qu'une tâche de classification des signaux reconstruits est plus adaptée.

1.2.3 Détection de l'impact du COVID long sur l'insomnie

Nous collectons et mettons à disposition de la communauté un jeu de données nommé COVISLEEP. Celui-ci comporte des polysomnographies de personnes ayant développé une insomnie chronique après infection du COVID, et de personnes souffrant d'insomnie chronique mais n'ayant pas été infectées par le virus. Nous montrons que des caractéristiques de macro-architecture du sommeil ne suffisent pas à séparer les deux populations. Nous comparons divers modèles sur la tâche de classification de sommeil sur le jeu COVISLEEP et identifions que l'approche Sleep-Linear [174] obtient les meilleures performances. Nous l'utilisons ensuite pour prédire si des fenêtres de polysomnographies proviennent de patients infectés par le COVID long ou non. Nous illustrons la complexité de la tâche, notamment due à la forte variabilité entre les patients.

1.3 Organisation du manuscrit

Ce manuscrit est organisé en 6 chapitres.

Nous présentons dans le chapitre 2 les séries temporelles et les méthodes utilisées pour les classifier, en nous intéressant particulièrement aux approches basées sur des distances et aux approches basées sur la génération de caractéristiques. Nous détaillons les principales distances utilisées pour comparer les séries temporelles et soulignons les avantages et inconvénients de DTW. Nous présentons les réseaux de neurones, en nous intéressant particulièrement aux architectures adaptées aux séries temporelles, et les autoencodeurs. Enfin, nous étudions la tâche de classification de sommeil, que nous utilisons pour valider nos travaux.

Dans le chapitre 3, nous présentons les principales méthodes d'approximation de DTW pour rendre l'algorithme plus rapide ou différentiable. Nous détaillons notre proposition d'approximation de DTW, l'architecture de l'autoencodeur utilisé et les performances de notre approche pour comparer des signaux de sommeil.

Dans le chapitre 4, nous introduisons les approches usuelles pour compresser des signaux temporels. Nous utilisons les approximations de DTW développées dans le chapitre 3 pour proposer une approche permettant de compresser des signaux de sommeil en dégradant au minimum l'information utile pour discriminer les états du sommeil. Nous étudions les performances de la compression via la classification de sommeil.

Dans le chapitre 5, nous décrivons le projet PANDORE-IA, dont l'objectif est la détection de l'impact du COVID long sur l'insomnie. Nous décrivons le jeu de données que nous avons collecté, ainsi que les tâches de classification de sommeil et de détection de

COVID que nous avons étudiées.

Enfin, le chapitre 6 conclut le manuscrit et donne des pistes de travaux futurs.

Chapitre 2

Classification de séries temporelles : contexte et cas particulier du sommeil

Sommaire

2.1 Classification de séries temporelles 25
2.1.1 Méthodes par modèles 25
2.1.2 Méthodes par distances 26
2.1.3 Méthodes par création de caractéristiques 30
2.1.4 Conclusion 48
2.2 Classification de sommeil 49
2.2.1 Principaux jeux de données 49
2.2.2 Métriques 52
2.2.3 Architectures de modèles et comparaisons 53
2.3 Conclusion et discussion 59

Classification de séries temporelles : contexte et cas particulier du sommeil

Une série temporelle $x \in \mathbb{R}^{N \times K}$ est une suite ordonnée d'échantillons $x_i \in \mathbb{R}^K$, $i \in [1, N]$, obtenus au cours du temps. Il existe ainsi un grand nombre de domaines utilisant ces séries, comme la finance, la médecine ou l'ingénierie, à l'image de la banque de données UCR [44]. De fait, de nombreuses tâches, comme l'indexation, le groupement (*clustering*) ou la classification, sont réalisées sur ces séries. La classification en particulier consiste à déterminer la classe y (ou *label*) d'une série temporelle x, parmi un ensemble de classes déterminées. Dans le contexte de l'apprentissage automatique, on définit cette tâche comme supervisée, car le modèle apprend à réaliser la tâche en évaluant sa prédiction $\hat{y}$ par rapport à la vérité y connue. Le but de la tâche est donc d'obtenir un modèle $\zeta : \mathbb{R}^{N \times K} \to \{y\}$ qui maximise la probabilité de prédire la bonne classe $p(\hat{y} = y|x)$, avec $\hat{y} = \zeta(x)$. Diverses méthodes existent pour obtenir le modèle ζ permettant de bien réaliser la tâche.

On peut baser la classification sur le choix d'une métrique pertinente pour séparer les éléments en groupes ou les associer avec des éléments dont on connait la classe et inférer celle de la série en question. La capacité de la métrique à correctement associer les séries temporelles similaires selon la tâche voulue est alors le principal facteur permettant d'améliorer les performances de classification. Le choix le plus courant est la distance euclidienne. Cependant, dans le contexte des séries temporelles, l'algorithme Dynamic Time Warping (DTW) possède d'intéressantes propriétés permettant de mieux discriminer certaines séries : DTW permet de comparer des séries de tailles différentes, et n'est pas sensible au décalage temporel. Mais les défauts de l'algorithme, principalement sa complexité temporelle et sa non-différentiabilité, freinent son utilisation dans un contexte d'apprentissage bout-en-bout, incitant la volonté d'approximer l'algorithme DTW. On peut ainsi choisir de représenter DTW par une fonction dont on apprend les paramètres. Les méthodes et modèles utilisés sont alors largement dépendants du type et du volume de données temporelles traitées. Dans cette thèse, on choisit de se concentrer sur la classification de données de sommeil.

Ces données permettent de classifier les états du sommeil d'un patient automatiquement à partir d'enregistrements. Elles sont néanmoins volumineuses et peuvent donc nécessiter d'être compressées. Dans le cas de l'apprentissage automatique, cela consiste à obtenir un encodeur ϕ tel que la représentation encodée de l'entrée x, nommée z, soit de plus faible dimension que x. On veut pouvoir restaurer les données et donc aussi obtenir le décodeur ψ tel que le signal reconstruit $\hat{x}$ à partir de z par le décodeur soit aussi "proche" de l'entrée que possible.

Dans ce chapitre, nous présenterons la tâche de classification de séries temporelles en détaillant en particulier les approches basées sur les distances et celles utilisant l'apprentissage profond. Nous présenterons les métriques utilisées pour comparer des séries temporelles, et illustrerons les avantages de DTW, puis nous nous intéresserons aux architectures de réseaux utilisées pour représenter les séries temporelles. Enfin, nous

étudierons dans le détail un cas particulier de séries temporelles : les enregistrements de sommeil. Nous expliquerons la tâche de classification du sommeil en présentant les principaux jeux de données, les méthodes d'évaluation des modèles, ainsi que les architectures des réseaux qui obtiennent les meilleures performances.

2.1 Classification de séries temporelles

Les méthodes pour classifier les séries temporelles sont généralement séparées en 3 catégories, comme présenté par [1] et [78] : basées sur des modèles, basées sur des distances et basées sur des caractéristiques. La première méthode, peu courante, suppose que des séries de même classe sont générées par le même "modèle" sous-jacent, ou par des modèles proches selon une métrique. On classifie donc une nouvelle série x en cherchant le modèle m qui s'ajuste le mieux à x et en affectant la classe associée à une série générée par le modèle le plus proche de m. Les méthodes basées sur des distances définissent des métriques utilisées pour comparer les séries entre elles. Pour classifier une nouvelle série, on recherche la série la plus proche au sens de la métrique choisie et on lui affecte sa classe. Enfin, les méthodes basées sur des caractéristiques transforment les séries temporelles en vecteurs de caractéristiques, puis des algorithmes apprennent la classification. Cette transformation est fixe et en amont de la tâche dans le cas de classifieurs dits traditionnels, comme les SVM [25] ou les *Random Forest* [26]. Elle est à l'inverse apprise par des réseaux de neurones, qui modifient les paramètres des transformations faites aux données pour mieux classifier.

2.1.1 Méthodes par modèles

Les méthodes par modèles cherchent à apprendre les paramètres d'un modèle génératif pour chaque série temporelle x. Les séries sont ensuite comparées les unes aux autres via la distance entre les modèles utilisés pour les représenter. Ainsi, [157] utilise un modèle de Markov caché (*Hidden Markov Model*, HMM). Celui-ci comporte deux variables, des états cachés H et des observations S. À un temps donné t, l'observation $S^{(t)}$ ne dépend que de l'état caché $H^{(t)}$. Les états cachés forment une chaîne de Markov : $H^{(t)}$ ne dépend que de $H^{(t-1)}$. Une autre approche usuelle est d'utiliser des modèles auto-régressifs (les éléments de la représentation sont générés les uns après les autres, en utilisant tous les éléments déjà générés) : ainsi [9] représente les séries selon la taille d'une "run", définie par le nombre d'éléments consécutivement au-dessus, ou en dessous, de la moyenne de la série.

La classification peut alors être réalisée via l'algorithme de plus proche voisin (KPPV) par exemple. On sépare le jeu de données $\mathcal{D}$ en un jeu support et un jeu requête. On crée des modèles pour toutes les séries du jeu support. Ensuite, pour chaque série du jeu requête $x^{(r)}$, on crée le modèle correspondant $m^{(r)}$ et on recherche le modèle $m^{(s)}$ du jeu support le plus proche selon une métrique. On parle alors de 1-PPV. On affecte ensuite à $x^{(r)}$ la classe de la série $x^{(s)}$, correspondant à $m^{(s)}$.

Ces approches sont cependant généralement inférieures aux méthodes par distances et par création de caractéristiques en termes de performance de classification [10] et ne seront donc pas étudiées dans ce manuscrit. Nous concentrons ce panorama sur les

méthodes par distance et les méthodes basées sur des caractéristiques

2.1.2 Méthodes par distances

Le choix de la distance utilisée pour comparer les séries est cruciale à la méthode. Cependant, puisque les données temporelles peuvent être de tailles et de dimensions variables, présenter du bruit, du décalage temporel et de fortes différences entre les amplitudes des dimensions, le choix n'est pas évident.

Les métriques sont séparées en 2 catégories principales. Celles qui comparent la distance entre le ième point d'une série et le ième point d'une autre série, comme la distance euclidienne, et celles qui calculent un chemin d'alignement non linéaire entre les séries, comme la Dynamic Time Warping (DTW), introduite conjointement et indépendamment par [177] et [146].

2.1.2.1 Distance euclidienne

La mesure de distance la plus usuelle est la distance euclidienne (ED). Elle est calculée entre deux séries temporelles $x, x' \in \mathbb{R}^{N \times K}$ par :

$$d_{ED}(x, x') = \sqrt{\sum_{i=1}^{N} \sum_{k=1}^{K} (x_{ik} - x'_{ik})^2} \tag{2.1}$$

Dans le cas des séries temporelles, la distance euclidienne présente plusieurs inconvénients. Tout d'abord, elle impose que les 2 séries soient de même longueur, ce qui n'est pas toujours possible en fonction des applications. Elle est de plus fortement sensible au décalage temporel, puisqu'elle compare les séries échantillon par échantillon.

Ainsi, des métriques plus adaptées aux séries temporelles ont été proposées dans la littérature. Nous les présentons dans la suite.

2.1.2.2 Dynamic Time Warping

DTW est une technique d'alignement de séries temporelles utilisée pour mesurer leur similarité. L'objectif de l'algorithme est de calculer la similarité entre deux séries temporelles $x, x' \in \mathbb{R}^{N \times K}, \mathbb{R}^{M \times K}$ avec N et M longueurs respectives des séries et K le nombre de canaux (ou dimensions) des échantillons. Pour ce faire, on commence par chercher l'alignement optimal entre x et x' puis on calcule la distance euclidienne entre les échantillons alignés. On peut formuler le problème sous la forme suivante :

$$DTW(x, x') = \min_{\pi \in A(x,x')} \left(\sum_{(i,j) \in \pi} d(x_i, x'_j) \right) \tag{2.2}$$

avec π un chemin d'alignement, $d(x_i, x'_j)$ la distance entre x_i et x'_j et $A(x, x')$ l'ensemble des chemins possibles. Ainsi la matrice d'alignement selon π, A_π, est définie par :

$$(A_\pi)_{i,j} = \begin{cases} 1 \text{ si } (i,j) \in \pi \\ 0 \text{ sinon} \end{cases} \tag{2.3}$$

En pratique, on peut calculer la DTW sous l'angle de la programmation dynamique comme défini par [150]. On calcule d'abord la matrice de distance $D \in \mathbb{R}^{N \times M}$avec :

$$D(i,j) = d(x_i, x'_j) + min[D(i-1, j-1), D(i-1, j), D(i, j-1)] \tag{2.4}$$

et on a alors :

$$DTW(x, x') = \sqrt{D(N, M)} \tag{2.5}$$

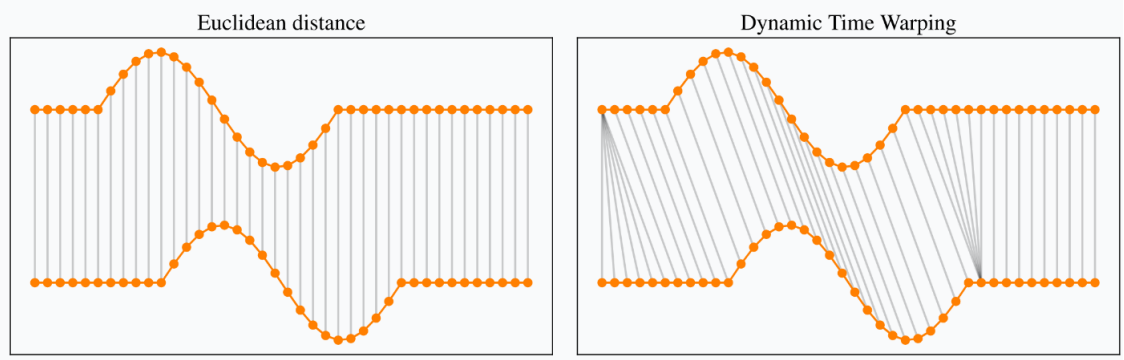


FIGURE 2.1 – Alignement de distance euclidienne et DTW. Par [168].

L'alignement effectué par la distance euclidienne et DTW est illustré figure 2.1.

La DTW n'est pas une distance. Bien qu'elle possède des propriétés basiques de métriques :

$$DTW(x, x') \geq 0 \ \forall x, x' \tag{2.6}$$

$$DTW(x, x) = 0 \ \forall x \tag{2.7}$$

elle ne respecte ni l'inégalité triangulaire ni l'identité des indiscernables [168]. Ce dernier point est dû au fait que la DTW est invariante au décalage temporel. En effet, si on définit $x_{(t)}$ comme x simplement décalée de t échantillons dans le temps, alors $DTW(x, x_{(t)}) = 0$. Cette propriété est un des avantages principaux de l'algorithme. L'autre point fort de DTW est sa capacité à comparer des séries de tailles différentes, ce que ne permettent pas les métriques usuelles comme la distance euclidienne.

2.1.2.3 Autres distances

D'autres méthodes d'alignement existent pour comparer les séries temporelles. La **Longest Common Subsequence** (LCSS) [72] cherche à obtenir la plus longue sous-séquence commune à deux séries temporelles $x \in \mathbb{R}^{N \times K}$ et $x' \in \mathbb{R}^{M \times K}$. Pour l'obtenir, on définit une matrice $D \in \mathbb{R}^{N \times M}$, puis $\forall i \in [1, N], j \in [1, M]$:

$$D(i,j) = \begin{cases} 0, & \text{si } i = 0 \text{ ou si } j = 0 \\ 1 + D(i-1, j-1), & \text{si } d(x_i, x'_j) \leq \epsilon \\ \max(D(i-1, j), D(i, j-1)) & \text{sinon} \end{cases} \tag{2.8}$$

avec $d(x_i, x'_j)$ la distance entre x_i et x'_j (en général on choisit la distance euclidienne) et ϵ un seuil de similarité. On a alors $LCSS(x, x') = D(N, M)$.

On définit la similarité basée sur cette distance par :

$$Sim_{LCSS}(x, x') = \frac{2 * LCSS(x, x')}{N + M} \tag{2.9}$$

Cela permet de borner les valeurs de similarités entre 0 et 1, et de comparer deux similarités entre elles, indépendamment de la longueur des séries.

Le **coefficient de Pearson** (pcc, ou ρ) mesure la corrélation entre deux séries temporelles x et x' de même longueur. Il est défini dans l'équation suivante [1] :

$$pcc(x, x') = \frac{\sum_{i=1}^{N} (x_i - \bar{x})(x'_i - \bar{x'})}{\sqrt{\sum_{i=1}^{N} ((x_i - \bar{x})^2) \sum_{i=1}^{N} ((x'_i - \bar{x'})^2)}} \tag{2.10}$$

où $\bar{x} = moyenne(x)$. $pcc(x, x')$ varie entre -1 et 1, 0 indiquant une corrélation nulle entre les deux séries, -1 une corrélation négative et 1 une corrélation positive. Si $x = x'$ alors $pcc(x, x') = 1$.

La **Move-Split-Merge** (MSM) [161] est proche d'autres techniques basées sur la distance d'édition, où la similarité est déterminée en utilisant un ensemble d'opérations pour transformer une série donnée en une autre série cible. MSM définit trois opérations : *move*, *split* et *merge*. L'opération *move* est similaire à une opération de substitution, remplaçant une valeur par une autre. L'opération *split* insère une copie d'une valeur juste après elle-même, et *merge* supprime une valeur si elle est suivie directement par une valeur identique. Nous détaillons le fonctionnement de MSM pour des signaux unidimensionnels [2] dans l'algorithme de programmation dynamique 1, extrait de [161]. La fonction de coût est définie par :

$$C(x_i, x_{i-1}, x'_j) = \begin{cases} c \text{ si } x_{i-1} \leq x_i \leq x'_j \text{ ou } x_{i-1} \geq x_i \geq x'_j \\ c + \min\left(|x_i - x_{i-1}|, |x_i - x'_j|\right) \text{ sinon.} \end{cases} \tag{2.11}$$

avec $c \in \mathbb{R}$.

La **Time Warping Edit Distance** (TWED) [105] intègre des éléments de DTW et de LCSS en permettant une déformation temporelle et en fusionnant la distance d'édition avec différentes normes *Lp*. Cette déformation, dénommée "rigidité", est paramétrée par une valeur ν. Alors qu'une fenêtre de déformation (comme Sakoe-Chiba [146], détaillée dans la section 3.2.1) limite la recherche du chemin dans DTW, la rigidité ajoute un coût multiplicatif à la distance entre les points jumelés. Si $\nu = 0$, on obtient une recherche DTW standard, tandis qu'avec $\nu = \infty$, on a une distance Euclidienne. La méthode TWED modifie les méthodes classiques d'insertion, de suppression et de mise en correspondance en utilisant "delete_a", "delete_b" et "match". Lorsqu'un élément est retiré de la première série pour correspondre à la seconde, c'est l'opération "delete_a" ; pour le cas inverse, c'est "delete_b". Lors de la mise en correspondance, on utilise une distance de type norme *Lp*, et en l'absence de correspondance, une pénalité fixe λ est ajoutée.

1. Ici, x et x' sont des signaux unidimensionnels, mais l'équation se généralise naturellement à des signaux à K dimensions.

2. Même remarque.

Algorithm 1 Programmation dynamique du calcul de la distance MSM

Entrée $x \in \mathbb{R}^N$ et $x' \in \mathbb{R}^M$ deux séries temporelles, $D \in \mathbb{R}^{N \times M}$ matrice nulle
Sortie $D(N, M)$

$D(1,1) = |x_1 - x'_1|$
for $i \in [2, N]$ **do** :
  $D(i,1) = D(i-1,1) + C(x_i, x_{i-1}, x'_1)$
end for
for $j \in [2, M]$ **do** :
  $D(1,j) = D(1,j-1) + C(x'_j, x_1, x'_{j-1})$
end for
for $i \in [2, N]$ **do** :
  for $j \in [2, M]$ **do** :

$$D(i,j) = \min(D(i-1,j-1) + \left|x_i - x'_j\right|, \\ D(i-1,j) + C\left(x_i, x_{i-1}, x'_j\right), \\ D(i,j-1) + C\left(x'_j, x_i, x'_{j-1}\right))$$

  end for
end for

2.1.2.4 Utiliser la distance pour classifier

De manière similaire aux méthodes basées sur des modèles, la classification peut être réalisée via un 1-PPV, en comparant directement les séries temporelles selon la métrique choisie.

En particulier, utiliser un 1-PPV basé sur DTW permet d'obtenir d'excellentes performances sur une large variété de jeux de données, comme illustré par [96]. Sur un ensemble de 76 jeux de données de séries temporelles de diverses sources, les 1-PPV basés respectivement sur MSM et DTW contrainte [3] obtiennent les meilleures performances en moyenne (rang moyen du classifieur de 4.85 / 12) par rapport à notamment LCSS, TWED et la distance euclidienne (ED). Dans une expérience similaire, [180] illustre que DTW performe mieux que la distance euclidienne et LCSS pour classifier des séries temporelles.

Ainsi, différentes distances existent pour comparer les séries temporelles : la classique distance euclidienne, qui compare les éléments un à un, DTW qui recherche à aligner les points des deux séries comparées, et d'autres distances qui modifient l'un des signaux pour obtenir l'autre, comme TWED et MSM. En l'utilisant comme métrique d'un 1-PPV, DTW permet d'obtenir de meilleurs scores de classification de séries temporelles que les autres métriques présentées précédemment. Ces résultats, combinés aux nombreuses approches permettant d'accélérer DTW (détaillées dans le chapitre suivant) justifient notre choix de concentrer notre étude sur cette métrique.

Néanmoins, la classification via métrique et 1-PPV nécessite de comparer les séries

3. Le chemin de recherche de DTW est usuellement contraint en pratique, pour améliorer les performances de l'algorithme et réduire la complexité. Les différentes contraintes seront détaillées dans le chapitre 3.

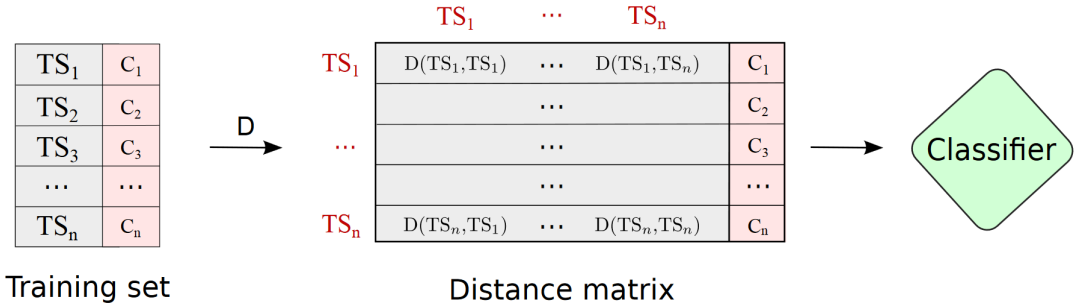


FIGURE 2.2 – Création de la matrice de caractéristiques via distance entre les séries temporelles. Extrait de [1]. Ici, les séries temporelles $x^{(i)}$ sont nommées TS_i, et les classes correspondantes C_i.

temporelles brutes, c'est-à-dire sans extraction de caractéristiques. Pour utiliser les capacités de représentation de plus puissants classifieurs, on peut utiliser les distances pour créer des caractéristiques [1].

2.1.3 Méthodes par création de caractéristiques

Plutôt que de traiter les séries temporelles directement, on les transforme en un ensemble de caractéristiques descriptives, qui captent l'essence et les motifs intrinsèques de la série. Ces caractéristiques sont ensuite utilisées comme entrées pour un classifieur traditionnel, tel qu'une machine à vecteurs de support (SVM), une forêt aléatoire ou un réseau de neurones. Les caractéristiques peuvent être générées par des transformations pré-déterminées (on parle de caractéristiques faites "à la main", ou *hand-crafted features*), ou apprises via des réseaux de neurones.

2.1.3.1 Hand-crafted features

De très nombreuses méthodes d'extraction de caractéristiques de séries temporelles existent [10], [57]. Les données audio sont par exemple transformées en représentation temps-fréquence via spectrogramme-mel [107], et des caractéristiques de signaux de sommeil sont créées en calculant leur densité spectrale (Sleep-Linear, [174]). Cependant, ces méthodes sont très largement dépendantes du jeu de données traité. Dans cette partie, nous nous concentrerons sur les méthodes basées sur les distances présentées précédemment pour étudier leur capacité de comparaisons des séries temporelles en tant que méthode de création de caractéristiques. Le principe de ces méthodes est de générer une matrice de caractéristiques via les distances entre les séries temporelles, puis de se servir de cette matrice pour apprendre la classification. On l'illustre figure 2.2, dans laquelle $TS_i = x^{(i)}$ indique la i-ème fenêtre de signal extraite de la série temporelle x. Ainsi, on crée une matrice $M \in \mathbb{R}^{n \times n}$ avec n le nombre de séries temporelles $x^{(i)}, \quad \forall i \in [1, n]$ dans le jeu de données. Ensuite, on remplit la matrice avec :

$$M(i,j) = D(x^{(i)}, x^{(j)}) \quad \forall i,j \in [1,n] \tag{2.12}$$

avec D la distance comparant les séries.

[82] utilise DTW pour générer les caractéristiques en comparant les distances entre les séries puis classifie via SVM (nous nommons cette approche SVM_{DTW_F}). Ils montrent que sur la banque de données UCR, SVM_{DTW_F} obtient de meilleures performances que la méthode 1-PPV basée sur DTW (que nous nommons 1-PPV_{DTW}). Sur les 47 jeux considérés, SVM_{DTW_F} est meilleur sur 31 jeux, moins bon sur 13 et équivalent sur 3. Ces méthodes sont cependant dépendantes des hyper-paramètres (paramètres d'algorithmes non appris), notamment le type de noyau choisi dans le cas du SVM.

Dans la suite de ce manuscrit, nous comparerons diverses métriques via la classification de séries temporelles. Pour limiter l'impact de facteurs extérieurs, comme le classifieur choisi, nous utiliserons donc la méthode 1-PPV basée sur une métrique.

Nous nous intéressons dans la suite de ce chapitre aux méthodes basées sur de l'apprentissage profond du fait des récentes performances du domaine [58], [57], [78].

2.1.3.2 Méthodes par apprentissage profond

L'apprentissage profond (*Deep Learning*, DL) est une sous-catégorie de l'apprentissage automatique (Machine Learning) qui utilise des architectures de réseaux de neurones artificiels profonds pour effectuer des tâches d'apprentissage. Le but est d'estimer un modèle $\zeta : x \rightarrow \hat{y}$, paramétré par des poids w, qui optimise $p(\hat{y} = y|x)$. Contrairement aux modèles de Machine Learning (ML) traditionnels, qui utilisent des caractéristiques, ou *features*, choisies par le *data scientist*, on applique les modèles de DL aux données "brutes", prétraitées mais sans extraction de caractéristiques. Le but est d'utiliser une combinaison de fonctions simples pour apprendre une tâche de bout en bout : de l'extraction implicite de caractéristiques jusqu'à la prédiction finale.

Apprentissage L'objectif de l'apprentissage est de trouver les poids w^* du modèle ζ qui maximisent un score. Cela nécessite un jeu de données $\mathcal{D}$, dont on extrait des entrées x et leurs *labels* correspondants y. Le modèle prédit une sortie $\hat{y} = \zeta(x; w)$ en fonction de ses poids w et de l'entrée x. Cette étape est appelée *forward pass*.

On compare $\hat{y}$ avec la vérité terrain y via une fonction d'erreur $\mathcal{L}$:

$$\mathcal{L} : \Big| \; y \in \mathbb{R} \times \hat{y} \in \mathbb{R} \;\; \longrightarrow \;\; \mathbb{R} \tag{2.13}$$

L'objectif est alors de minimiser la valeur de $\mathcal{L}$. La fonction d'erreur doit être différentiable pour permettre de mettre à jour les poids w via descente de gradient. Les choix usuels sont la Mean Squared Error (MSE) pour les tâches de régression (c'est-à-dire que y est une valeur réelle) et la Cross-Entropy (CE) pour les tâches de classification.

Celles-ci sont définies comme suit.

La MSE est une fonction de coût basée sur la distance euclidienne. Elle est définie pour deux séries $x, x' \in \mathbb{R}^{N \times K}$ par :

$$MSE = \frac{1}{N * K} \sum_{i=1}^{N} \sum_{k=1}^{K} (x_{ik} - x'_{ik})^2 \tag{2.14}$$

Elle présente de fait les mêmes inconvénients que la distance euclidienne pour la comparaison de séries temporelles : elle nécessite des séries de même longueur, et est sensible au décalage temporel.

La *Cross-Entropy* est définie par :

$$H(y,p) = -\sum_{i=1}^{Cl} y^{(i)} \log(p_i) \tag{2.15}$$

où Cl est le nombre de classes, $y^{(i)}$ est 1 si la classe i est la vraie classe et 0 sinon, et p_i est la probabilité prédite pour la classe i.

On peut alors réaliser l'apprentissage, dont l'objectif est le suivant :

$$w^* = \underset{w}{argmin} \sum_{x,y \in \mathcal{D}} \mathcal{L}(\zeta_w(x), y) \tag{2.16}$$

Cet apprentissage est basé sur l'algorithme de descente de gradient. Celui-ci consiste à itérativement mettre à jour les poids w_i^t à l'itération t pour minimiser l'erreur entre les valeurs prédites et les valeurs réelles. Ainsi, on modifie les poids de la manière suivante :

$$w_i^{t+1} = w_i^t - \Delta w_i^t \tag{2.17}$$

avec :

$$\Delta w_i^t = \eta \frac{\delta \mathcal{L}(\zeta_{w^t}(x), y)}{\delta w_i^t} \tag{2.18}$$

avec η le pas de descente de gradient, indiquant la vitesse de mise à jour des poids. Via la règle de dérivation en chaîne, on applique cette opération à toutes les couches du réseau, et on met ainsi à jour tous les poids du réseau, en commençant par la fin du réseau et en remontant jusqu'à l'entrée. C'est la rétro-propagation du gradient (*backpropagation*).

Le processus d'apprentissage est résumé dans l'algorithme 2. Il consiste à extraire des sous partie des matrices de données et de labels appelés *batch*, dont la taille est un hyper-paramètre. Pour chaque *batch*, le modèle prédit la sortie $\hat{y}$, qui est comparée aux valeurs cibles y pour mettre à jour les poids du modèle. Cette opération est répétée jusqu'à atteindre un critère, qui est en général un nombre d'itérations de mise à jour des poids. Par exemple, il est courant d'arrêter l'apprentissage après un nombre défini d'*epochs*. Une epoch indique avoir itéré sur tout le jeu d'entraînement une fois.

Naturellement, avoir un modèle performant sur le jeu d'apprentissage n'est pas suffisant. On étudie les performances d'un modèle sur un jeu de test, c'est-à-dire des données avec lesquelles le modèle n'a jamais mis à jour ses paramètres. Lors de l'apprentissage, on cherche à réduire deux catégories d'erreur :

— Le biais qui est l'erreur systématique que l'on peut observer entre les prévisions d'un modèle et les valeurs réelles qu'il tente de prédire. Il est généralement causé par la simplification excessive du modèle, qui le rend incapable de capturer les nuances des données d'entraînement. Quand le biais est majoritaire, on dit que le modèle sous-apprend.

Algorithm 2 Optimisation des poids d'un modèle par descente de gradient stochastique

Entrée Modèle ζ avec des poids aléatoires w^0, taux d'apprentissage η, jeu de données $\mathcal{D}$
Sortie Modèle ζ avec des poids optimisés

```
t ←0
while critère d'arrêt non atteint do :
    for X_batch, Y_batch ∈ D do :
        Δw_i^t ← 0
        for x, y ∈ X_batch, Y_batch do :
            ŷ ← ζ(x, w^t)
            Δw_i^t ← Δw_i^t + δL(y,ŷ)/δw_i^t
        end for
        Δw_i^t ← η Δw_i^t / card(X_batch)
        t ← t + 1
        w_i^t ← w_i^{t−1} − Δw_i^{t−1}
    end for
end while
```

— La variance qui est l'erreur aléatoire qui se produit lorsque le modèle est trop complexe et surajuste les données d'entraînement. Cela signifie que le modèle s'adapte trop étroitement et ne généralise pas bien à de nouvelles données. On dit que le modèle sur-apprend. Ce phénomène est très courant quand le volume de données est faible.

Réduire les deux erreurs en même temps est souvent difficile. Ce problème est connu sous le nom de dilemme biais-variance [60]. Il est très difficile de connaître à priori la complexité suffisante du modèle pour ne pas sous-apprendre. Ainsi, la pratique usuelle est de choisir des modèles complexes, et de lutter contre le sur-apprentissage.

Une façon de lutter contre le sur-apprentissage est de contrôler l'erreur commise sur le jeu de validation grâce à l'arrêt anticipé.

L'arrêt anticipé, ou *early stopping*, illustré dans la figure 2.3, consiste à arrêter l'entraînement d'un modèle dès que sa performance sur le jeu de validation commence à se dégrader, plutôt que de continuer à entraîner le modèle jusqu'à la convergence complète. L'early stopping permet ainsi de choisir le modèle qui généralise le mieux aux nouvelles données en arrêtant l'entraînement à un moment où la performance sur le jeu de validation est optimale. Cela permet également d'économiser du temps de calcul en évitant de poursuivre un apprentissage qui n'apporterait pas plus d'amélioration significative sur de nouvelles données. Cependant, cette technique repose sur l'hypothèse que les distributions sous-jacentes des données dans ces différents ensembles sont identiques, c'est-à-dire $P_{train}(X,Y) = P_{validation}(X,Y) = P_{test}(X,Y)$. On dit alors qu'on satisfait l'hypothèse d'indépendance et d'identiquement distribuée (i.i.d.). Cette hypothèse est souvent tenable lorsque les ensembles de données sont suffisamment grands et que les échantillons sont tirés de manière aléatoire. Cependant, si cette condition n'est pas satisfaite, l'optimalité des performances sur l'ensemble de validation ne ga-

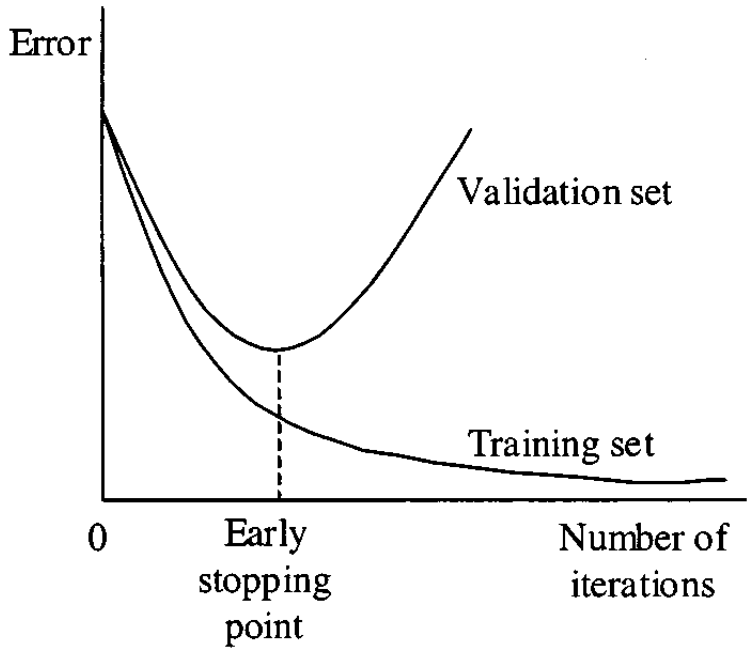


FIGURE 2.3 – Early stopping basé sur l'erreur sur le jeu de validation. Extrait de [61].

rantit pas nécessairement l'optimalité des performances sur l'ensemble de test. Or, dans le cas des données de santé, la séparation des jeux de données est réalisée par patient (aucune donnée d'un patient dans le jeu de test ne doit être dans les jeux d'entraînement et de validation). Ainsi, le tirage des échantillons dans les jeux n'est pas strictement aléatoire, et l'arrêt anticipé n'offre pas toujours des améliorations de performance.

Pour obtenir de bonnes performances sur la tâche souhaitée, il faut que l'architecture du réseau de neurones permettent de représenter efficacement les signaux d'entrée. Nous détaillons dans la suite diverses architectures de réseaux de neurones.

Perceptron Le premier réseau de neurones, le perceptron, fut proposé par Rosenblatt en 1958. On l'illustre figure 2.4. La sortie o est obtenue via une somme pondérée de l'entrée $x = [x_1, ..., x_n]$ par les poids $w = [w_1, ..., w_n]$ avant qu'un terme de biais b soit ajouté, c'est-à-dire qu'on obtient :

$$o = \sum_{i=1}^{n} x_i * w_i + b \tag{2.19}$$

On applique ensuite une fonction d'activation A à o. Dans le cas du perceptron, elle est très simple :

$$\hat{y} = A(o) = \begin{cases} 0, & \text{si } o < 0 \\ 1, & \text{sinon} \end{cases} \tag{2.20}$$

Cette simplicité limite la capacité d'apprentissage du perceptron à des fonctions linéaires.

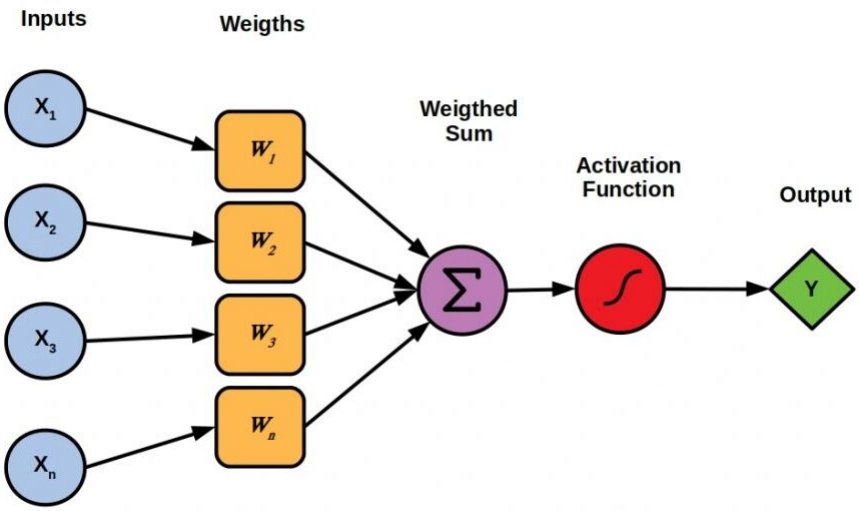


FIGURE 2.4 – Exemple de perceptron. [4]

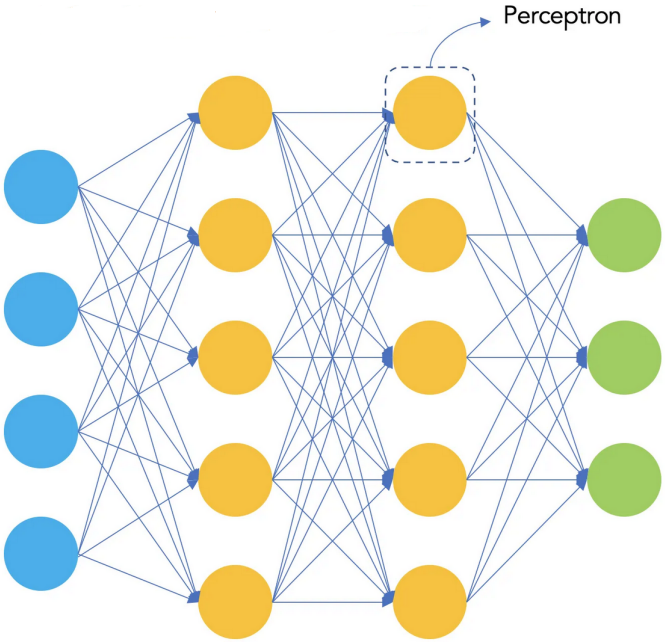


FIGURE 2.5 – Multi Layer Perceptron. Extrait de [5]

Perceptron multi-couches Le perceptron multi-couches (*Multi Layer Perceptron*, MLP) se compose d'une succession de couches de neurones, disposées les unes après les autres. Les connexions entre les neurones ne se font qu'entre couches adjacentes. Tous les neurones de deux couches adjacentes sont connectés entre eux : le MLP est donc un réseau *fully connected* (FCN). Ces couches sont nommées denses ou linéaires. Les connexions se font de plus toujours de l'entrée vers la sortie : le MLP est un réseau dit *feed forward* (FCN). Un MLP constitué de deux couches est illustré dans la figure 2.5. Le nombre de couches représente alors la profondeur du réseau. En augmentant le nombre de couches, on construit des caractéristiques de différents niveaux sémantiques. On parle d'apprentissage profond (*Deep Learning*, DL).

Des fonctions d'activation sont utilisées à la sortie de chaque couche. En choisissant des fonctions d'activation non linéaires, le MLP est capable d'apprendre des fonctions non linéaires. Ces fonctions sont appliquées à la sortie o d'une couche de réseau. Les fonctions d'activation les plus communes sont ReLU [114], définie par :

$$ReLU(o) = \max(0, o) \tag{2.21}$$

Tanh [79], définie par :

$$tanh(o) = \frac{e^o - e^{-o}}{e^o + e^{-o}} \tag{2.22}$$

Softmax [64], définie par :

$$softmax(o) = \frac{e^o}{\sum_j^J e^{o_j}} \forall i \in \{1..J\} \tag{2.23}$$

et *Sigmoid*, définie par :

$$sigmoid(o) = \frac{1}{1 + e^{-o}} \tag{2.24}$$

Le choix de la fonction d'activation est généralement décidé par essai/erreur, mais des tendances existent, comme illustré par [115] : la ReLU est majoritairement utilisée pour les couches du réseau servant à l'extraction de caractéristiques, tandis que la softmax permet de convertir les sorties du réseau en probabilités.

2.1.3.3 Types de réseaux pour les séries temporelles

En définissant des architectures particulières pour les réseaux de neurones, nous pouvons faciliter la représentation de certains types de données, et donc l'apprentissage de diverses tâches. Nous décrivons les architectures adaptées aux séries temporelles dans la suite.

Réseaux de neurones récurrents. Les réseaux de neurones récurrents (RNN) [141] sont un type de réseaux de neurones dans lesquels les neurones sont interconnectés de manière récurrente. Ce principe permet de simuler une « mémoire » grâce à un état interne maintenu tout au long du processus d'apprentissage. Cela rend les RNN particulièrement adaptés pour traiter les données temporelles.

4. https://starship-knowledge.com/wp-content/uploads/2020/10/Perceptrons-1024x724.jpeg
5. https://medium.com/swlh/pysparks-multi-layer-perceptron-classifier-on-iris-dataset-dcf70d553cd8

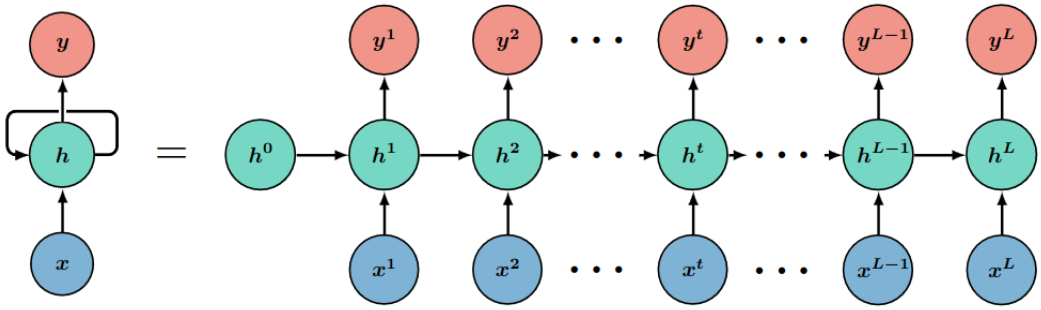


FIGURE 2.6 – RNN, compressé et déplié. Par [39].

Le fonctionnement d'un RNN est illustré figure 2.6. Ici, x^t représente le signal d'entrée à l'étape t. L'état caché h^t est calculé à partir de l'état caché précédent h^{t-1} et du vecteur d'entrée x^t. Ainsi, la sortie finale y^L dépend directement de tous les états cachés h et de tous les échantillons de l'entrée x. La *backpropagation* est effectuée en remontant la chaîne à l'envers. Ceci créé une difficulté : puisqu'il faut un état caché h par échantillon de x, l'impact des premiers états sur le gradient est limité, et ils ne sont que peu mis à jour par la *backpropagation* : c'est le phénomène de disparition du gradient (*vanishing gradient*).

Pour limiter ce problème, [73] ont introduit les réseaux *Long Short-Term Memory* (LSTM). Le fonctionnement d'une cellule LSTM est illustré figure 2.7. La mémoire d'un LSTM est représentée par une cellule c et un état caché h. Les informations transitent par 3 portes :

— La porte d'entrée (*Input Gate*) quantifie l'information à apporter par la nouvelle entrée x^t à la mémoire c^t.

— La porte d'oubli (*Forget Gate*) régule la quantité d'information à supprimer de la mémoire c^{t-1} pour passer à c^t

— La porte de sortie (*Output Gate*) sert à calculer h^t à partir de c^t. C'est la sortie de la cellule.

Cette architecture lourde aide à mieux représenter la série en mémoire sans en oublier des parties. C'est cependant au prix de plus de paramètres, et donc de temps d'apprentissage et d'inférence plus longs.

Pour tenter de combiner les capacités de mémoire des LTSM et la relative légèreté des RNN, [35] ont introduit les *Gated Recurrent Unit* (GRU). Ceux-ci ne contiennent pas de cellules c, et les fonctions de la porte d'entrée et d'oubli du LSTM sont combinées dans une porte de mise à jour z. Une autre porte de *reset* r contrôle la quantité d'information oubliée pendant la mise à jour à l'état suivant.

Il n'existe cependant pas de consensus sur le meilleur choix parmi les 3 architectures précédentes. Ces réseaux sont de plus assez lents dans le cas d'une tâche de génération : puisqu'il faut se servir de l'élément y^{t-1} pour générer y^t, on doit générer les séries élément par élément.

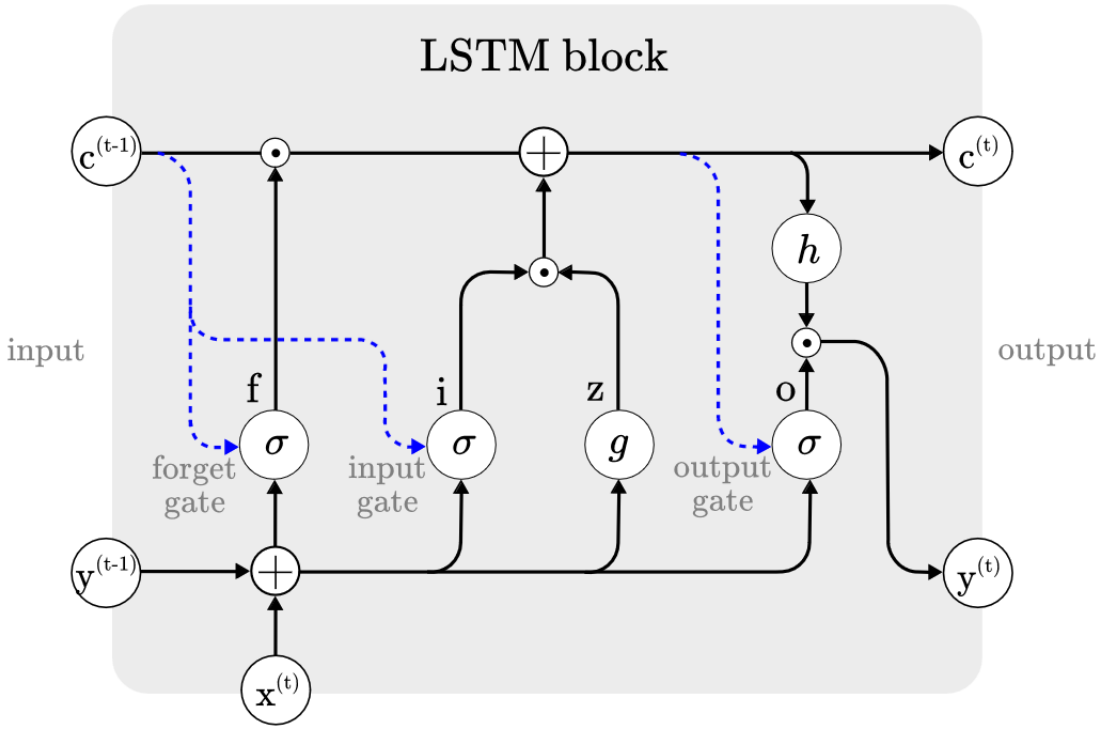


FIGURE 2.7 – Cellule standard LSTM. Par [175].

Réseaux de neurones à convolution Ainsi, bien que les réseaux de neurones récurrents soient efficaces pour modéliser les dépendances séquentielles au sein de séries temporelles, ils présentent divers inconvénients. Notamment, ils sont vulnérables au problème de disparition du gradient et peuvent être computationnellement coûteux en raison de leur grand nombre de paramètres. Une alternative pour pallier ces problèmes consiste à employer des réseaux de neurones à convolution.

Ils sont un type de réseau de neurones basés sur un enchaînement de couches de convolution et de couches *pooling*, avant une couche dense finale. Introduit par [92] avec LeNet-5, les réseaux de convolutions sont aujourd'hui incontournables pour les tâches liées aux images ou aux sons. Par exemple, dans le cas des images, une couche de convolution utilise un ensemble de k noyaux de convolution $W_1, W_2, ..., W_k$, chacun ayant une taille $h \times w \times nc$, où h, w et nc sont respectivement la hauteur, la largeur et le nombre de canaux de x. La sortie de la couche de convolution est une carte de caractéristiques c, où chaque pixel $c_{i,j}$ correspond à une convolution de x avec un noyau W_k. La convolution est effectuée en déplaçant le noyau sur x avec un certain pas, appelé pas de convolution (ou *stride*) s, et en multipliant les deux matrices élément par élément, puis en sommant les résultats.

Les couches de *pooling* sont des couches qui permettent de réduire la dimensionnalité des représentations. Elles fonctionnent de manière similaire à la couche de convolution, en balayant un filtre sur toute l'entrée. Cependant, contrairement à la couche de convolution, le filtre de *pooling* n'a pas de poids, mais applique une fonction d'agrégation aux représentations des entrées. Les plus communes sont le *MaxPooling*, qui ne conserve que la valeur maximale dans la fenêtre du filtre, et le *average pooling* qui pro-

duit la moyenne des valeurs de la fenêtre. Ces deux opérations permettent de réduire le nombre de paramètres du modèle, et donc d'accélérer l'apprentissage.

Alors que les couches de pooling jouent un rôle crucial dans la réduction de la dimensionnalité et l'accélération de l'apprentissage, d'autres techniques sont également utilisées pour améliorer la performance des modèles de réseaux de neurones. Une de ces techniques est le dropout [160], qui offre une méthode efficace pour mitiger le sur-apprentissage.

Ainsi, **le dropout** consiste à nullifier certains neurones du réseau pendant l'entraînement. À chaque itération, chaque neurone possède une probabilité p d'avoir son vecteur de poids w nul. Le but est de forcer le modèle à utiliser tous les neurones, et de ne pas sur-utiliser l'information qui proviendrait de certains d'entre eux. La réduction significative de l'erreur de classification grâce au dropout [117] en fait un bloc présent dans la très grande majorité des architectures de réseaux de neurones.

Puisque les noyaux de convolution sont appliqués localement, le dropout usuel peut dégrader le modèle [59]. Ainsi, le Spatial Dropout [171] a été introduit pour contourner cet inconvénient. Il consiste à ignorer intégralement une dimension de la matrice de caractéristiques pour empêcher le réseau d'utiliser ces connexions.

Bien que conçus pour l'image, les réseaux à convolutions sont performants dans de nombreuses tâches liées aux séries temporelles, et particulièrement la classification [75]. Le Temporal Convolutional Network (TCN), introduit par [11] est proposé comme une famille d'architectures simples et puissantes, servant de *baseline* compétitive. L'architecture est définie pour satisfaire deux critères : les convolutions sont causales et le modèle prend des entrées x de longueur variable, et produit des sorties z de même taille que x, comme les RNN. Pour ce faire, les couches de convolution sont remplacées par des convolutions dites dilatées, suivant [173]. Celles-ci permettent de prendre en compte des entrées plus éloignées les unes des autres dans chaque opération de convolution, intégrant ainsi des informations à partir d'un champ réceptif plus large. Pour une série $x \in \mathbb{R}^N$ et un filtre $f : \{0, \dots, k-1\} \to \mathbb{R}$, la convolution dilatée est définie par :

$$F(s) = (x *_q f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-q \cdot i} \tag{2.25}$$

avec k la taille du filtre de convolution et q le facteur de dilatation, $*_q$ indiquant ainsi la convolution dilatée de facteur q. Ainsi, en utilisant des convolutions dilatées et en les empilant, un réseau de neurones peut prendre en compte des informations provenant d'une grande zone de la série temporelle d'entrée sans augmenter le nombre de couches ou les coûts computationnels. Avec $q = 1$, la convolution dilatée équivaut à la convolution classique. L'architecture d'une convolution dilatée est illustrée dans la figure 2.8.

Le deuxième élément caractéristique du TCN est la présence de connexions résiduelles (*residual connection*, ou *skip connection*), proposé par [69], dans le but de conserver et de "rappeler" au modèle la série initiale. La connexion résiduelle consiste à faire subir une transformation $\mathcal{F}$ à l'entrée x d'une partie du réseau, puis d'ajouter (ou

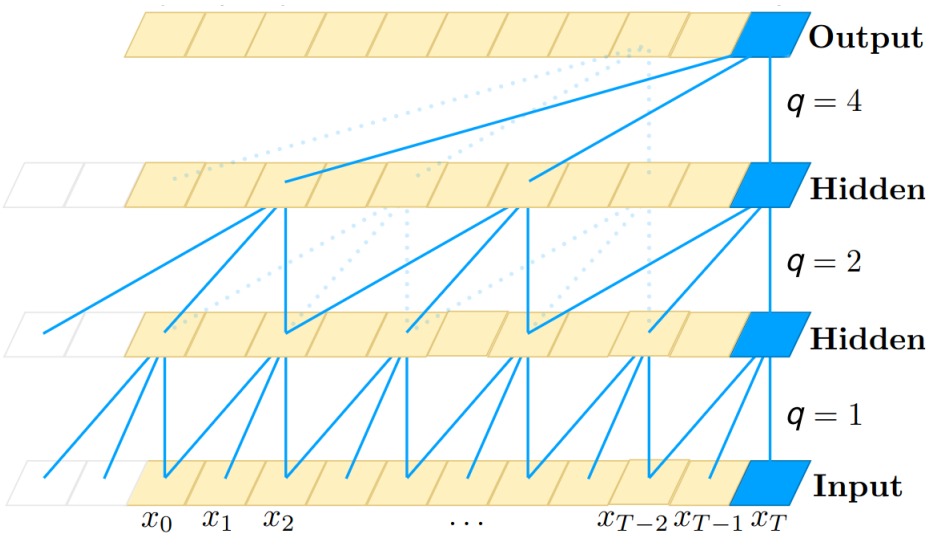


FIGURE 2.8 – Convolution dilatée avec q=[1,2,4] et k = 3. Extrait de [11]

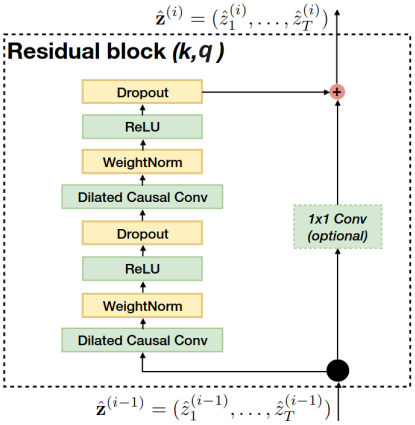


FIGURE 2.9 – Skip connection. Extrait de [11], k est la taille du filtre de convolution, et q le facteur de dilatation.

concaténer) x avec $\hat{z} = \mathcal{F}(x)$. La sortie o du bloc est alors : $o = Activation(x + \mathcal{F}(x))$.

Un bloc du TCN est présenté figure 2.9. L'entrée de la couche i, $\hat{z}^{(i-1)}$, est transformée par une couche de convolution dilatée causale. Les poids sont mis à l'échelle par *Weight Normalisation* [148]. Cette opération consiste à transformer les poids d'une couche de réseau par : $w = g\frac{v}{||v||}$, où $v \in \mathbb{R}^k$ est le vecteur des poids de la couche, et g un scalaire. La fonction d'activation ajoutant la non-linéarité est ReLU, puis une opération de dropout aide à réduire le sur-apprentissage. Du côté de la connexion résiduelle, une convolution 1d est appliquée si $\hat{z}^{(i-1)}$ et $\hat{z}^{(i)}$ ne sont pas de même dimension.

Les réseaux convolutionnels ont ainsi plusieurs avantages face aux réseaux récurrents. Les sorties sont toutes générées en même temps, ce qui permet d'utiliser les capacités de parallélisation des cartes graphiques (GPU). Cela évite de plus de rétro-propager dans la direction temporelle de la série, et donc limite les problèmes de disparition ou explo-

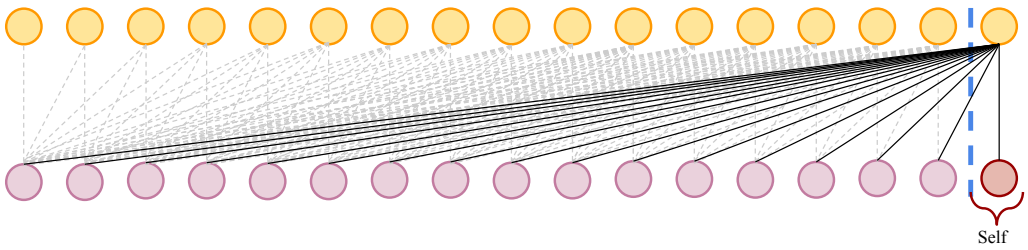


FIGURE 2.10 – Illustration du mécanisme d'attention sur séries temporelles. Par [95].

sion du gradient. Enfin, [11] illustre que sur diverses tâches de représentation de séries temporelles (classification, imputation de valeurs manquantes, sélection d'entrée), le TCN obtient des performances significativement supérieures à celles obtenues par des réseaux récurrents, indépendamment du type de cellule récurrente utilisée.

Néanmoins, une troisième architecture, particulièrement adaptée au texte, peut aussi être utilisée sur les séries temporelles.

Réseaux à attention. En 2017, [176] révolutionne le traitement automatique du texte en introduisant le *Transformer*. Ce modèle est basé sur le mécanisme d'attention. Son but est de se concentrer sur les parties pertinentes d'une séquence en attribuant des poids aux différents éléments en fonction de leur importance relative.

Son fonctionnement, illustré dans la figure 2.10, suit les étapes suivantes :

- Pour chaque élément d'une séquence, le modèle calcule trois vecteurs distincts à partir de leur représentation continue (*embeddings*) : les vecteurs de requête (Q), de clé (K) et de valeur (V). Ces vecteurs sont obtenus en multipliant les *embeddings* par des matrices de poids apprises (W). Les vecteurs Q , K et V sont utilisés pour déterminer l'importance relative des éléments de la séquence.
- Le modèle calcule ensuite les scores d'attention en effectuant un produit scalaire entre Q et K. Les scores d'attention reflètent la similarité entre les éléments de la séquence et sont utilisés pour pondérer l'importance relative de chaque élément.
- Les scores d'attention sont ensuite normalisés à l'aide d'une fonction *softmax*, qui transforme les scores en probabilités comprises entre 0 et 1.
- Chaque vecteur V est ensuite multiplié par son score d'attention correspondant. Les vecteurs d'attention pondérés ainsi obtenus mettent en évidence les éléments importants de la séquence en fonction de leur contexte.
- Enfin, les vecteurs d'attention pondérés sont additionnés pour obtenir le vecteur de sortie de la couche d'attention. Ce vecteur de sortie combine les informations des éléments de la séquence en fonction de leur importance relative.

Dans le cas des *Transformer*, le mécanisme d'attention est utilisé dans les couches d'auto-attention multi-tête. L'auto-attention indique que les calculs de score d'attention sont faits en comparant une séquence à elle-même. L'attention multi-tête signifie que plusieurs têtes d'attention fonctionnent en parallèle, chacune se concentrant sur

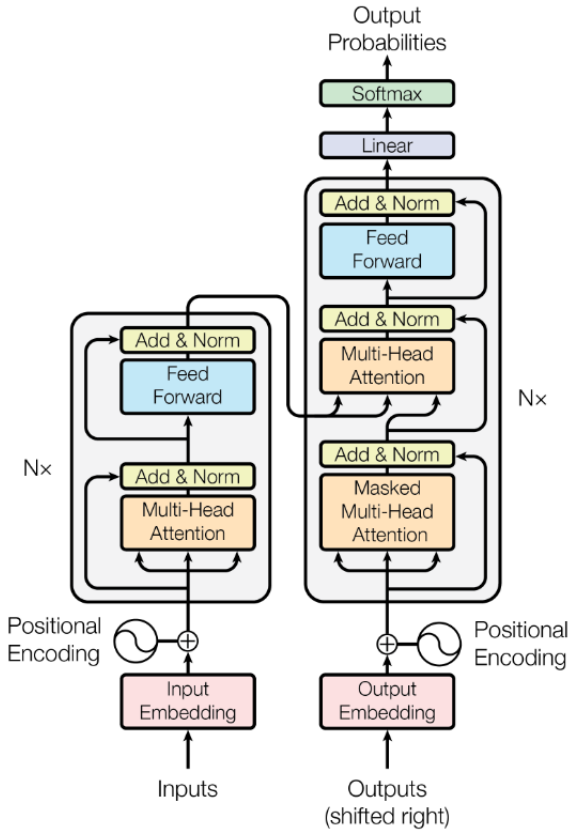


FIGURE 2.11 – Architecture du *Transformer*. Par [176].

différents aspects des relations entre les éléments de la séquence. Les résultats de chaque tête d'attention sont ensuite concaténés et transformés par une matrice de poids apprise pour produire le vecteur de sortie final de la couche d'attention multi-tête.

Dans sa globalité, le *Transformer* se base sur l'architecture encodeur-décodeur. On l'illustre figure 2.11.

L'encodeur est chargé de traiter la séquence d'entrée et de générer une représentation continue pour chaque élément de la séquence. Il est composé de plusieurs couches identiques, chacune ayant deux sous-modules principaux : la couche d'auto-attention multi-tête et la couche de "réseau de neurones à propagation avant" (*Feed Forward Neural Network*, FFNN). La couche d'auto-attention permet au modèle de considérer les relations entre tous les éléments de la séquence d'entrée, tandis que la couche FFNN ajoute de la non-linéarité au modèle. Comme l'encodeur, le décodeur est également composé de plusieurs couches identiques, chacune ayant trois sous-modules principaux : la couche d'auto-attention multi-tête (ciblée sur la séquence de sortie), la couche d'attention multi-tête (ciblée sur la sortie de l'encodeur), et la couche FFNN. Le décodeur génère la séquence de sortie un élément à la fois, en utilisant les informations de l'encodeur et en tenant compte des éléments précédemment générés.

Le texte pouvant être vu comme une série temporelle, le *Transformer* a donc été utilisé pour classifier, inférer ou générer des séries, à l'image de [15] par exemple, qui utilise un

réseau basé sur des *Transformer* pour imputer les valeurs manquantes de manière plus précise que l'architecture [30] basée sur des réseaux récurrents. Il permet une meilleure représentation temporelle des entrées que la convolution, puisque l'attention peut se porter à toute la série, et est plus rapide que les RNN car facilement parallélisable. Il est de plus un peu plus interprétable grâce aux probabilités obtenues en sortie de couche d'attention.

Ainsi, les RNN, CNN et réseaux à attention sont conjointement utilisables pour résoudre diverses tâches liées aux séries temporelles. Choisir entre ces méthodes nécessite donc de spécifier le type de séries temporelles traitées. Dans cette thèse, nous appliquerons les méthodes présentées à des données de santé et plus particulièrement des enregistrements de sommeil qui, comme la majorité des données médicales, sont peu labellisées.

2.1.3.4 Le manque de données labellisées

Le cas des données de santé. Utiliser des données médicales oblige à étendre le problème de généralisation de plusieurs façons. Tout d'abord, la généralisation à des patients différents est un élément obligatoire à prendre en compte. Si les plus célèbres jeux de données d'image (comme ImageNet [50]) ou de texte (comme Common Crawl[6]) proviennent de nombreuses sources, les jeux de données de médecine sont souvent restreints à un nombre de patients limité ($n < 100$). Ainsi, le modèle peut avoir des difficultés à généraliser ses connaissances à de nouveaux patients car il est biaisé par le faible nombre de patients sur lesquels il a appris. Il est donc important de garder des patients uniquement dans le jeu de test pour évaluer cette capacité de généralisation.

Ensuite, un modèle idéal traitant des données de santé doit être capable de réaliser sa tâche indépendamment de la méthode d'obtention des données. En effet, les capteurs, méthodes d'acquisition et experts (qui créent les labels nécessaires à l'apprentissage) sont très souvent différents d'un hôpital à l'autre. Ainsi, il est délicat de créer un modèle capable de bien généraliser sur un jeu de données différent. On dit d'un modèle capable de bien généraliser à des nouveaux patients qu'il est robuste.

Si le volume des données accessibles est conséquent, obtenir les labels correspondants est souvent coûteux, puisque cela nécessite le travail d'experts qualifiés. De fait, les jeux de données sont souvent restreints et le modèle risque le sur-apprentissage. Nous présentons dans la suite diverses techniques permettant de lutter contre le sur-apprentissage.

L'augmentation de données consiste à appliquer de légères modifications aux données d'entraînement pour augmenter artificiellement le volume de données disponible. On peut utiliser la technique sur toutes les données, ou sur certaines classes sous-représentées, pour équilibrer la distribution entre les classes dans le jeu d'apprentissage et éviter que le modèle ne soit biaisé vers les classes majoritaires. Le choix des types d'augmentations est délicat : il faut que les données soient suffisamment

6. https://commoncrawl.org/

modifiées pour éviter le sur-apprentissage, mais qu'elles conservent l'information permettant de les classifier. Si pour l'image le choix et l'intérêt des augmentations ont été étudiées en profondeur ([124]), leur application aux données EEG est plus récente [138]. Les augmentations permettant la plus haute amélioration des performances varient selon les tâches et les jeux de données, néanmoins les augmentations modifiant la représentation temps-fréquence des EEGs sont les plus couramment utilisées pour améliorer la classification du sommeil. Notamment, [138] montre qu'utiliser les augmentations *FTSurrogate*, qui modifie la phase des canaux des signaux en représentation de Fourier par une valeur aléatoire (bornée), *SmoothTimeMask*, qui masque une partie du signal, *TimeReverse*, qui inverse l'axe temporel des séries, et *GaussianNoise*, qui ajoute un bruit gaussien à la série, permet d'améliorer le score de classification de séries temporelles quand le volume du jeu de données est faible : seulement 1% du volume du jeu de données initial est conservé par [138] pour illustrer cette amélioration. Ces augmentations sont illustrées figure 2.12.

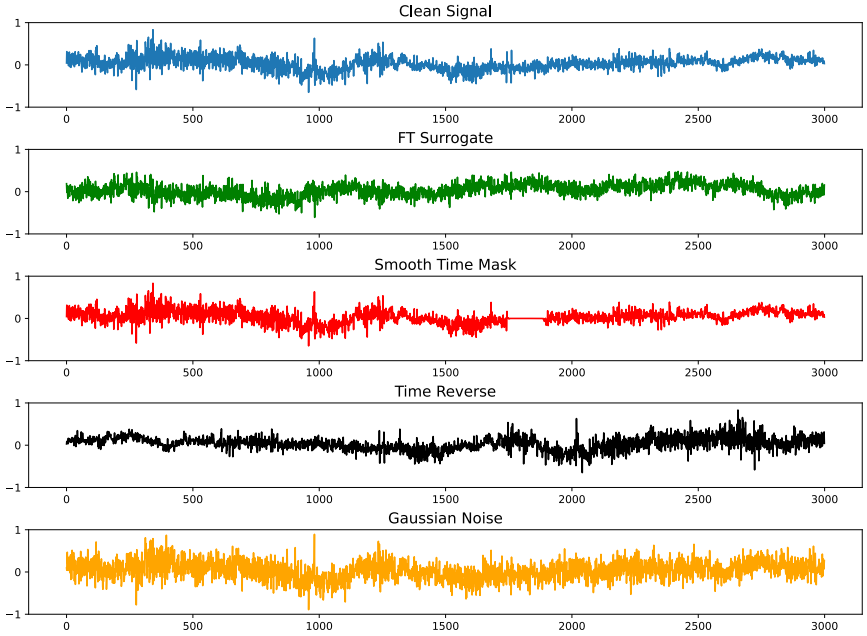


FIGURE 2.12 – Effet de certaines augmentations sur un signal EEG

Transfer Learning et autoencodeurs. Avec l'augmentation de données, le *transfer learning* est une des méthodes les plus directes pour lutter contre le sur-apprentissage dû à la difficulté d'obtenir de gros volumes de données labellisées. Cela consiste à d'abord pré-apprendre un modèle m_0 sur un jeu de données D_0 via une tâche T_0, souvent autosupervisée (qui ne nécessite pas de labels ajoutés humainement). On utilise ensuite les poids d'un certain nombre de couches de m_0 pour initialiser le modèle cible m_c. On apprend enfin m_c à réaliser la tâche cible T_c sur le jeu cible D_c. On dit alors qu'on ajuste, ou *finetune* le modèle m_c. Selon les cas pratiques, on peut ne pas mettre à jour certaines couches de m_c (on dit que les couches sont gelées). Cette technique

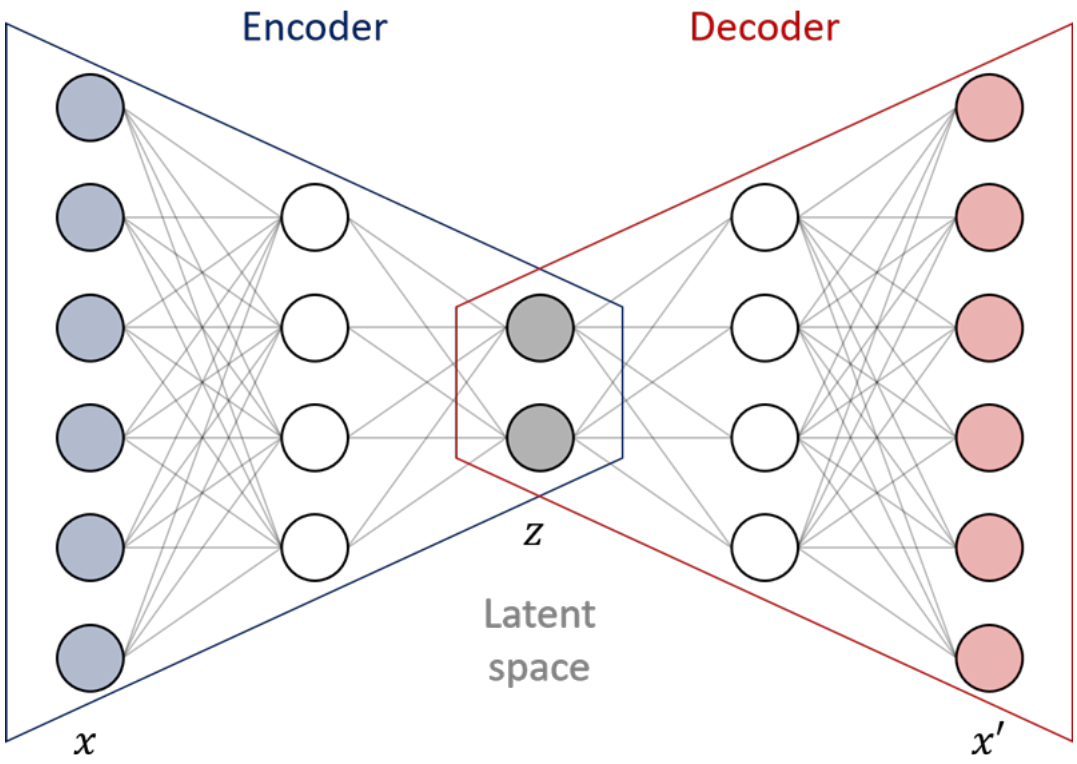


FIGURE 2.13 – Représentation d'un autoencodeur. [7]

est beaucoup utilisée dans tous les domaines de l'apprentissage profond [178], [190]. Si $m_0 = m_c$ et $D_0 = D_c$, on peut réaliser les deux opérations conjointement. On parle alors d'apprentissage multitâche, dont le but est d'utiliser T_0 pour régulariser T_c.

Le pré-apprentissage est usuellement réalisé via un autoencodeur, comme illustré par [119], qui utilisent un autoencodeur pour initialiser les poids d'un réseau récurrent apprenant à prédire la prochaine note d'une partition.

Un autoencodeur [12] est un type de réseau de neurones dont la sortie est de même dimension que l'entrée. Il est composé de deux parties : un encodeur ϕ, qui projette l'entrée en une représentation latente, et un décodeur ψ, qui reconstruit l'entrée à partir de cette représentation. Une représentation d'un autoencodeur est donnée dans la figure 2.13. La capacité de représentation d'un autoencodeur est entièrement dépendante de celle de ϕ et ψ. Si les deux opérations sont linéaires (avec des fonctions d'activation aussi linéaires) alors l'autoencodeur se comporte comme l'algorithme d'analyse en composantes principales (ACP) [14].

Ainsi, un autoencodeur cherche à reproduire l'entrée modifiée du modèle. Il ne nécessite pas de labels ajoutés humainement, et est donc particulièrement adapté pour pré-apprendre un modèle. [144] utilise notamment un autoencodeur basé sur des couches LSTM pour pré-apprendre un modèle capable de prédire le futur de séries temporelles,

7. https://blog.engineering.publicissapient.fr/2020/12/09/detection-de-fraudes-par-autoencodeur-variationnel-entraine-sur-google-ai-platform/

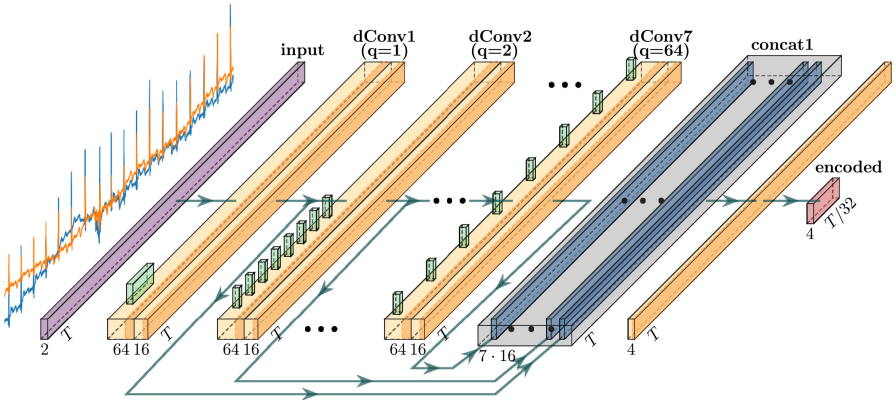


FIGURE 2.14 – Architecture de l'encodeur du TCN-AE. Extrait de [170]. Dans chaque bloc TCN, dConv indique une couche de convolution dilatée de taux q et de taille de noyau 64. Une couche de convolution 1×1 réduit ensuite la dimension des caractéristiques à 16. Les différentes représentations obtenues par les blocs TCN sont concaténées dans la partie concat1.

et [189] utilise le préapprentissage pour améliorer les performances en classification. Dans ce cas, la tâche de préapprentissage est du débruitage : on ajoute un bruit gaussien au signal et on demande au modèle de retrouver le signal propre.

Les autoencodeurs sont aussi utilisés pour de nombreuses autres applications. Dans le cas des séries temporelles, ils peuvent être utilisés pour la détection d'anomalies. Cette tâche consiste à distinguer des éléments "fautifs" (défauts de capteurs, signaux de santé de personnes malades, etc) parmi un jeu de donnée d'éléments "normaux". Quand la tâche est supervisée, on peut utiliser un autoencodeur comme extracteur de caractéristiques, avant un module de classification ([99]). Quand ce n'est pas le cas, on utilise le fait que les autoencodeurs apprennent la distribution normale des signaux. Ainsi, les signaux dont la reconstruction est la plus erronée sont probablement les signaux anormaux ([63], [143]).

Pour obtenir un autoencodeur capable de bien représenter les signaux temporels, il faut construire l'encodeur et le décodeur en utilisant les architectures de réseaux adaptées. Ainsi, [170] introduisent le TCN-AE, autoencodeur basé sur le TCN présenté précédemment.

Le TCN-AE est donc composé d'un encodeur et d'un décodeur, dont les architectures sont similaires entre elles. L'encodeur est représenté figure 2.14. Il est composé d'une pile de blocs TCN, similaires à celui représenté dans la figure 2.9. Les différentes représentations de la série ainsi obtenues sont ensuite concaténées pour former une pile de caractéristiques. Une couche de convolution 1×1 est appliquée pour réduire la dimensionnalité de ces caractéristiques. Enfin, une couche de *MaxPooling* [152] réduit la longueur du signal. Les principaux hyperparamètres du réseau sont les taux de dilatation q (qui contrôlent l'écart entre chaque entrée utilisée dans la convolution), le nombre de filtres utilisés dans les couches convolutionnelles (qui contrôlent le nombre de caractéristiques extraites à chaque couche), et la taille du noyau de convolution (qui contrôle la taille de la fenêtre de convolution).

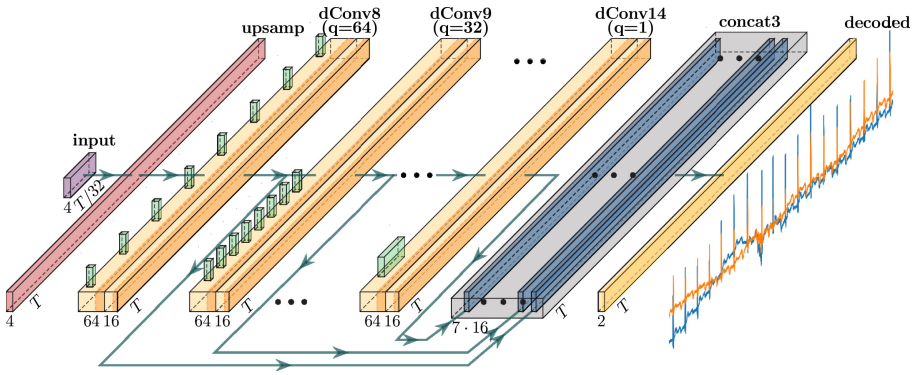


FIGURE 2.15 – Architecture du décodeur TCN-AE. La couche *upsamp* ré-échantillonne l'entrée du décodeur de sorte que le signal retrouve sa taille initiale (avant d'être encodé). Ensuite le fonctionnement est analogue à l'encodeur, avec différentes convolutions dilatées *dconv* puis une concaténation des représentations (*concat3*).

Le décodeur est présenté figure 2.15. Il est fait de manière analogue à l'encodeur, en inversant simplement l'ordre des taux de dilatation q. Le signal encodé est rétabli à sa taille d'origine via un sur-échantillonnage, puis le réseau essaie de retrouver le signal original.

Cette architecture permet au TCN-AE de réaliser l'état de l'art dans la détection d'anomalies de signaux de santé ECG, performant notamment mieux que des autoencodeurs basés sur des LSTM ([102], [169]). Pour ces raisons, nous utiliserons cet autoencodeur dans la suite de nos expérimentations.

Enfin, l'autoencodeur peut être utilisé pour compresser des signaux aussi bien en tâche de préapprentissage qu'en tâche finale. En choisissant simplement $z = \phi(x)$ de dimension inférieure à x, les signaux sont compressés dans l'espace latent. L'autoencodeur apprend à résoudre la tâche suivante :

$$\min_{\phi,\psi} \mathcal{L}(x, \psi(\phi(x))) \tag{2.26}$$

avec $\mathcal{L}$ la fonction de coût. Ainsi, la qualité de la reconstruction effectuée par l'autoencodeur dépend de la fonction de coût choisie. Le choix usuel est la MSE, par exemple dans [145], [16], [51], trois travaux traitant de la compression de séries de santé. Néanmoins, [143] démontre l'avantage d'utiliser des fonctions de coût intégrant des termes spécifiques aux données temporelles. Cette problématique sera détaillée et étudiée dans les chapitres suivants.

Normalisation et standardisation. La distance entre deux séries dépend directement de la valeur des échantillons. Ainsi, pour maintenir la stabilité numérique de la métrique, et donc du gradient mis à jour via cette métrique dans le contexte d'apprentissage d'autoencodeurs, il faut mettre à l'échelle les valeurs des séries temporelles. Pour ce faire, il existe deux méthodes principales : la normalisation et la standardisation.

La normalisation, également appelée mise à l'échelle min-max, projette les valeurs entre 0 et 1. On l'obtient via l'équation 2.27. Cette transformation préserve les relations de

rang entre les valeurs. Elle permet de borner les métriques et donc de les mettre à l'échelle de la même façon. Il faut cependant faire attention aux valeurs aberrantes : si $min(x) << mean(x)$, ou $mean(x) << max(x)$, alors la majorité des valeurs seront projetées autour de 0.5, ce qui peut gêner l'apprentissage. Dans ce cas, nous proposons de borner les valeurs aberrantes. Nous détaillons cette approche dans le chapitre suivant.

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)} \tag{2.27}$$

La standardisation, également appelée mise à l'échelle z-score, projette les valeurs de telle sorte qu'elles aient une moyenne $\mu = 0$ et un écart type $\sigma = 1$. Cette transformation préserve les distances entre les valeurs. Elle est définie par l'équation 2.28. La standardisation est la méthode la plus courante dans l'apprentissage profond, car elle permet d'accélérer la convergence, notamment en limitant les problèmes de disparition du gradient [62].

$$x_{stand} = \frac{x - mean(x)}{std(x)} \tag{2.28}$$

Néanmoins, la normalisation est intéressante dans le cas d'apprentissage avec des métriques, c'est-à-dire quand la sortie du modèle $\hat{x} = \zeta(x, w)$ est de même dimension que l'entrée x du modèle. On compare alors $y = \zeta(x, w)$ et $\hat{x}$ via une métrique, qui sert de fonction de coût. C'est notamment le cas des autoencodeurs.

2.1.4 Conclusion

Un grand nombre d'approches a été proposé pour comparer et classifier les séries temporelles. Notamment, les méthodes basées sur des distances permettent de classifier via un 1-PPV en cherchant simplement la série la plus proche. Pour passer outre les limites de la distance euclidienne, c'est-à-dire sa sensibilité aux décalages temporels et son incapacité à comparer des séries de tailles différentes, diverses métriques, incluant DTW, LCSS ou TWED ont été proposées. DTW, utilisée comme métrique d'un 1-PPV, offre de très bonnes performances de classification [96] [180]. La métrique est cependant de complexité quadratique en temps et non différentiable, ce qui nuit à son utilisation. Plus récemment, et pour traiter des volumes de données plus importants, des méthodes basées sur des réseaux de neurones ont été proposées. Elles permettent d'apprendre une représentation des séries temporelles permettant de mieux les classifier. S'il est classique d'utiliser les réseaux récurrents, il est préférable de choisir des architectures convolutionnelles pour représenter les séries temporelles, peu importe la tâche finale souhaitée, du fait de leurs meilleures performances et de leur vitesse d'exécution [11]. Des architectures basées sur des couches *Transformer* ont plus récemment été proposées, mais leur faible amélioration des performances ne justifie pas toujours leur utilisation, au vu de leur lenteur et de la complexité de leur apprentissage. Enfin, les tâches nécessitant de comparer les séries modifiées aux séries d'entrées, comme la compression, sont réalisées par des autoencodeurs utilisant les architectures représentant bien les séries temporelles comme le TCN. Cependant, ils se basent sur la MSE pour mettre à jour les poids du modèle, plutôt que sur les métriques adaptées mentionnées précédemment. Nous proposons donc d'approximer DTW, puis de l'utiliser

comme fonction de coût pour mieux apprendre les autoencodeurs sur les séries temporelles. Pour effectuer nos expérimentations et valider l'utilité de ce choix de fonction de coût, nous choisissons de nous intéresser aux enregistrements de sommeil, dont la classification automatique est très importante pour aider les experts à diagnostiquer des anomalies plus efficacement.

2.2 Classification de sommeil

La classification automatique du sommeil est une tâche essentielle dans le domaine de la recherche sur le sommeil et la santé. Elle consiste à identifier et à classer les différents stades du sommeil, généralement à partir de signaux polysomnographiques (PSG), qui peuvent inclure des enregistrements électroencéphalographiques (EEG), électrooculographiques (EOG) et électromyographiques (EMG). La classification du sommeil a des implications importantes pour l'évaluation de la qualité du sommeil, la détection des troubles du sommeil et la compréhension des mécanismes neurobiologiques du sommeil [123]. La tâche nécessite beaucoup de travail à des experts et a donc un coût important. Elle est, de plus, assez fastidieuse. Ainsi, automatiser cette tâche présente un intérêt significatif.

On définit la classification du sommeil comme une spécification de la classification de série temporelle, où l'entrée $x^{(i)} \in \mathbb{R}^{N \times K}$ est la i-ème fenêtre de 30 secondes extraite d'un enregistrement de sommeil, et le label correspondant $y^{(i)}$ peut prendre 5 valeurs selon l'American Academy of Sleep Medicine [74] (AASM) :

- l'éveil (Wake W), caractérisé par des ondes bêta (>12Hz) et alpha (8-12Hz).
- Le sommeil paradoxal (REM) marqué par des ondes bêta et des mouvements oculaires rapides.
- L'état de sommeil profond N1, caractérisé par la transition entre les ondes bêta et thêta (4-8 Hz).
- L'état de sommeil profond N2, marqué par des fuseaux de sommeil (*spindles*, salves de signaux oscillatoires provenant du thalamus et représentés dans l'EEG [49]).
- L'état de sommeil profond N3, caractérisé par des ondes delta (0.5-4 Hz).

Les analyses sont généralement réalisées sur des jeux de données publics, pour simplifier les comparaisons entre les différentes méthodes. Nous les présentons dans la suite de ce chapitre.

2.2.1 Principaux jeux de données

Dans cette partie, nous présentons les différents jeux de données habituellement utilisés pour apprendre à classer automatiquement les états du sommeil.

2.2.1.1 SHHS

Le Sleep Heart Health Study (SHHS), proposé par [132] et mis à jour par [186], est une cohorte d'études massive sur les effets cardiovasculaires et autres effets de la respiration

perturbée pendant le sommeil. Les participants à cette étude présentent différentes maladies, notamment des troubles cardiovasculaires et pulmonaires.

Il est courant dans la littérature ([153], [56]) de ne garder que deux sous parties du dataset : les études SHHS1 et SHHS2, qui correspondent à l'enregistrement d'une nuit de sommeil d'un groupe de patients à deux dates différentes. Ainsi, le jeu de données comprend deux enregistrements de sommeil pour 5791 patients différents. En particulier, la grande majorité des études se concentre sur le jeu de données SHHS1 ([159], [153], [125]). De fait, nous utilisons uniquement ce jeu, que nous nommons SHHS. À chaque fenêtre de 30 secondes est associé un label parmi l'éveil (Wake W), le sommeil paradoxal REM, divers stades de profondeur de sommeil : N1, N2, N3, N4 et des marqueurs d'activité : MOVEMENT, and UNKNOWN, suivant les recommandations Rechtschaffen & Kales (R&K [7]). Les enregistrements sont échantillonnés à 125 Hz, donc les fenêtres font chacune une taille $N = 3750$. Le jeu de données est préparé en enlevant les fenêtres MOVEMENT et UNKNOWN car elles ne contiennent pas d'information utile à la tâche de classification du sommeil. Ensuite, les états N3 et N4 sont fusionnés suivant les recommandations de l'AASM [74]. Les données SHHS ainsi obtenues sont fortement déséquilibrées, comme indiqué table 2.1. Cela illustre notamment la difficulté à identifier l'état N1, et pose la question de l'intérêt de garder la classe au lieu de la fusionner avec N2. Néanmoins, pour faciliter les comparaisons avec les autres méthodes, nous conserverons les pré-traitements décrits précédemment.

Les données sont de plus multidimensionnelles car comportant 2 canaux EEGs, 2 EOGs, 2 EMGs, 1 ECG (électrocardiogramme), une pléthysmographie inductive respiratoire à deux canaux, des données de capteur de position, des données de capteur de lumière, des données d'oxymètre de pouls et des données de capteur de flux d'air. Tous les canaux ne sont pas utilisés par la littérature : [130], [129] exploitent les canaux EEG, EOG et EMG, mais [153], [93] utilisent seulement le channel EEG C4-A1 pour des performances équivalentes. Du fait de l'accélération des calculs permise par l'utilisation d'un seul channel, nous ne conserverons que le channel C4-A1 dans nos expériences sur SHHS.

2.2.1.2 SleepEDF

Le jeu de données SleepEDF [83] est un autre jeu de données contenant des enregistrements de sommeil labellisés. Il contient 197 PSG répartis en 2 sous jeux : *Sleep-Cassette* (SC) et *Sleep-Telemetry* (ST).

Le jeu SC contient 153 polysomnographies correspondant à 78 patients. Ce jeu est parfois nommé SC-EDF-78 dans la littérature. Pour chaque patient sauf 3, 2 enregistrements de nuit sont disponibles comprenant 2 signaux EEG (Fpz-Cz et Pz-Cz), 1 EOG, 1 EMG et pour certains patients un capteur de respiration et de température. Tous les signaux sont bruts et échantillonnés à 100 Hz, sauf l'EMG qui est filtré et ré-échantillonné à 1 Hz. Il est commun dans la littérature de nommer SC-EDF-20 le sous jeu de données ne comportant que les patients de 0 à 19, qui fut publié avant le jeu complet.

Le jeu ST contient 44 enregistrements de 22 patients. Pour chaque patient, 2 nuits de

Dataset	Wake	N1	N2	N3	REM	Total des fenêtres
SHHS	23 %	4 %	44 %	14 %	15 %	5.421.338
SC-EDF-78	34 %	11 %	35 %	7 %	13 %	195.168

TABLE 2.1 – Répartitions des labels selon les classes et nombre total de labels des jeux de données SHHS-1 et SC-EDF-78.

sommeil sont disponibles : une après la consommation de temazepam, un médicament contre l'insomnie, et une après la consommation d'un placebo. Les mêmes canaux, sans les capteurs de respiration et de température, sont disponibles.

Le jeu SC-EDF-78 étant le plus couramment utilisé dans la littérature ([165], [129], [131]), nous utiliserons exclusivement ce jeu dans nos expérimentations que nous nommons généralement SleepEDF-78.

Comme pour le jeu SHHS, les informations de sommeil ont été renseignées par des experts suivant les recommandations R&K. Le même traitement est donc usuellement appliqué pour transformer les labels. Le jeu de données résultant est de nouveau déséquilibré comme détaillé table 2.1, avec une large domination des classes Wake et N2.

De manière similaire à SHHS, des approches multidimensionnelles pour résoudre la tâche de classification de sommeil, utilisant les canaux EEG et EOG ([66], [130]), ou EEG, EOG et EMG ([174]) existent et obtiennent d'excellentes performances. Néanmoins, d'autres travaux se limitent au canal Fpz-Cz ([164], [131]) pour des performances similaires. Nous présenterons dans le tableau 2.3 un comparatif des méthodes à l'état de l'art sur les jeux de données SHHS et SleepEDF-78, ce qui conforte notre décision d'utiliser des séries unidimensionnelles. Dans le cas de SleepEDF-78, nous conservons le canal Fpz-Cz, suivant la littérature mentionnée précédemment.

2.2.1.3 Autres jeux de données

De nombreux autres jeux de données de sommeil sont accessibles : entre autres, MASS [116] regroupe les enregistrements de polysomnographie de 200 patients obtenus dans 3 hôpitaux différents. DOD-O [67] est composé des enregistrements de 55 patients atteints d'apnée du sommeil. Les annotations sont réalisées par 5 experts de différents centres du sommeil, permettant de comparer les performances des modèles à celles d'humains. Un jeu de données analogue, DOD-H, est obtenu sur des volontaires sains. ISRUC [85] regroupe 3 jeux de données : le premier contient un enregistrement d'une nuit de 100 patients, mêlant individus sains, individus souffrant de problèmes de sommeil et individus utilisant des traitements contre l'insomnie. Le deuxième comporte deux enregistrements de 8 patients et le troisième un enregistrement de 10 individus. Nous concentrons cependant nos analyses sur les jeux de données SHHS et SleepEDF-78, qui sont les plus couramment utilisés dans la littérature.

2.2.2 Métriques

On évalue les modèles de classification du sommeil via des indicateurs de performance nommés métriques. Le détail du fonctionnement des métriques les plus courantes dans la littérature est donné dans cette section. La prédiction d'un modèle ζ pour un signal x est notée $\hat{y} = \zeta(x)$, et sa classe réelle est notée y.

2.2.2.1 Accuracy

L'*accuracy* est simplement définie par le rapport entre le nombre de bonnes prédictions et le nombre total de prédictions. Elle permet une évaluation rapide des performances d'un modèle mais ne prend pas en compte le possible déséquilibre entre les classes.

2.2.2.2 Macro F1

Pour prendre en compte ce déséquilibre, on utilise le score F1. Pour une classe c donnée, on sépare les prédictions du modèle en 4 cas :

- *true positive* (TP) : le modèle prédit qu'un signal x est de classe c (on définit cette prédiction par $\hat{y} = c$) et la classe réelle de l'élément est effectivement c (on note $y = c$)
- *false positive* (FP) : $\hat{y} = c$ et $y \neq c$
- *false negative* (FN) : $\hat{y} \neq c$ et $y = c$
- *true negative* : $\hat{y} \neq c$ et $y \neq c$

On obtient ensuite la *precision* via l'équation 2.29, mesurant parmi toutes les prédictions de classes c faites par le modèle, combien le sont effectivement :

$$Precision_c = \frac{TP}{TP + FP} \tag{2.29}$$

Et le *recall* via l'équation 2.30, qui évalue combien d'éléments de classe c ont été prédits parmi tous les éléments dont la vraie classe est c :

$$Recall_c = \frac{TP}{TP + FN} \tag{2.30}$$

On peut ensuite calculer pour chaque classe c le score $F1_c$ selon l'équation 2.31. Cela permet d'avoir une compréhension des performances du modèle classe par classe.

$$F1_c = 2 * \frac{precision_c * recall_c}{precision_c + recall_c} \tag{2.31}$$

On obtient un score global pour le modèle, nommé score macro F1 (MF1) via l'équation 2.32 avec Cl le nombre de classes.

$$MF1 = \frac{\sum_{c=1}^{Cl} F1_c}{Cl} \tag{2.32}$$

Ainsi, chaque classe a le même impact sur le score final, peu importe son nombre d'éléments. Cette métrique pénalise donc fortement les modèles qui ne prédisent pas du tout une ou plusieurs classes.

2.2.2.3 Kappa de Cohen

Le coefficient Kappa de Cohen [37] (κ) est une mesure statistique qui évalue l'accord entre 2 annotateurs y_1 et y_2. Il est souvent utilisé pour mesurer la fiabilité des annotations et la cohérence inter-évaluateurs dans des tâches telles que la classification manuelle d'images, l'annotation de texte ou la classification d'état du sommeil, à l'image de [67], qui utilise la métrique pour comparer le score de classification d'experts et de modèles. Il prend en compte la possibilité que les évaluateurs se mettent d'accord par hasard et compare l'accord observé entre les évaluateurs à l'accord attendu en supposant que les évaluations sont indépendantes et aléatoires. On le définit ainsi :

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{2.33}$$

p_o est la probabilité observée d'accord entre les évaluateurs, définie par :

$$p_o = \frac{1}{N} \sum_{i}^{Cl} n_{ii} \tag{2.34}$$

avec Cl le nombre de classes et n_{ii} les cas où les deux annotateurs ont prédit la classe i. p_e est la probabilité attendue d'accord si les évaluations étaient aléatoires et indépendantes, et s'exprime par :

$$p_e = \sum_{i}^{Cl} \frac{n_{1i} * n_{2i}}{N^2} \tag{2.35}$$

avec n_{1i} et n_{2i} les prédictions de la classe i par les annotateurs 1 et 2 respectivement et N le nombre total d'annotations. Le score étant symétrique, on choisit y_1 comme vérité terrain et y_2 les prédictions du modèle pour la classification du sommeil.

Le score Kappa de Cohen est particulièrement utile pour mesurer l'accord de prédictions de différents modèles pour paramétrer des ensembles (joindre les prédictions de plusieurs modèles sur le même jeu de données pour améliorer les performances) ou pour mesurer l'accord inter-annotateurs. Si l'*accuracy* ne permet pas d'évaluer la capacité du modèle à correctement prédire les classes moins représentées, le score macro F1 permet lui d'obtenir ces informations. Nous utilisons donc uniquement l'*accuracy* et le score macro F1 dans nos expériences.

2.2.3 Architectures de modèles et comparaisons

Depuis 2017, la grande majorité de la recherche sur la classification automatique de sommeil s'est concentrée sur l'utilisation de réseaux de neurones profonds. Les premières approches ([128], [127]) utilisaient simplement une fenêtre donnée de signal et tentaient de prédire le label correspondant. C'est cependant insuffisant, car les états du sommeil n'évoluent pas fréquemment : sur le jeu SleepEDF-78, le label $y^{(t+1)}$ est identique au label $y^{(t)}$ 89.8 % du temps. On présente de plus dans le tableau 2.2 les pourcentages d'évolution des états en t et $t+1$. Comme indiqué précédemment, tous les états restent très majoritairement identiques à la fenêtre suivante. Les autres transitions ne sont pas aléatoires : par exemple, bien que les états W et N2 représentent plus

de la moitié du jeu (table 2.1), si l'état t est W alors $t+1$ ne sera N2 que dans 0.24% des cas, et inversement dans 1.49% des cas. Ainsi, utiliser les fenêtres précédentes est utile pour prédire la classe de la série en cours.

$y^{(t)} \setminus y^{(t+1)}$	W	N1	N2	N3	REM
W	94.63%	5.01%	0.24%	0.01%	0.12%
N1	8.87%	72.42%	15.83%	0.04%	2.85%
N2	1.49%	2.66%	90.82%	3.83%	1.19%
N3	1.00%	0.58%	18.72%	79.65%	0.10%
REM	1.82%	2.77%	1.31%	0.004%	94.08%

TABLE 2.2 – Pourcentage de transition entre les différents stades du sommeil du jeu de données SleepEDF-78

Cette idée est renforcée par le fait que les experts utilisent plusieurs fenêtres autour de la fenêtre $x^{(t)}$ pour prendre leur décision [126]. En utilisant ces propriétés, des méthodes exploitant une ou deux fenêtres précédant la fenêtre courante et une fenêtre suivant la fenêtre courante ont permis d'améliorer l'état de l'art. Elles sont basées sur des CNN (SorsNet [159] [8], [32], U-Sleep [125]), des RNNs ou une combinaison des deux ([101]). Cependant, ces approches n'exploitent pas assez de fenêtres pour bien représenter les liens passés ou futurs.

Les approches modernes sont représentées par des modèles séquences vers séquences [126]. Ces modèles reçoivent L fenêtres continues et prédisent en même temps les L labels correspondants. Ils sont composés de 2 sous modèles principaux : un encodeur de caractéristique $Fc : x^{(1:L)} \rightarrow h^{(1:L)}$, dont le rôle est de projeter $x^{(1:L)}$ pour en extraire les caractéristiques utiles, et un encodeur temporel $Ft : h^{(1:L)} \rightarrow z^{(1:L)}$, qui utilise les L représentations $h^{(1:L)}$ pour affiner les projections. Ainsi chaque $z^{(l)}$ pour $l \in [1..L]$ contient l'information de la projection de son signal brut affiné par les projections des signaux proches temporellement. Un classifieur (souvent une couche linéaire) estime ensuite les prédictions $\hat{y}^{(1:L)}$.

Beaucoup de modèles suivant ce paradigme obtiennent d'excellentes performances. Par exemple, le DeepSleepNet [165], SleepEEGNet[113], le TinySleepNet[164] et IITNet [153] modélisent Fc par un CNN et Ft par un RNN. En particulier, DeepSleepNet est illustré figure 2.16. L'encodeur Fc est composé de deux branches de CNN contenant des couches avec différents paramètres (par exemple, la première couche a une taille de filtre de la moitié de la fréquence d'échantillonnage Fs) du signal, un noyau de taille 64 et un pas de taille $Fs/16$. Les représentations extraites sont concaténées puis envoyées dans l'encodeur temporel, composé d'une double couche LSTM bidirectionnelle. Une *skip connection* comportant une couche linéaire permet de conserver la représentation extraite de l'encodeur de caractéristiques. Des couches de *dropout* permettent de limiter le sur-apprentissage.

SimpleSleepNet [67], SeqSleepNet [129] et RobustSleepNet [66] utilisent des réseaux récurrents pour l'encodeur de caractéristiques, avec un module d'attention pour le Ro-

8. Nous présentons en détail l'architecture du SorsNet dans la section 3.3.3.1

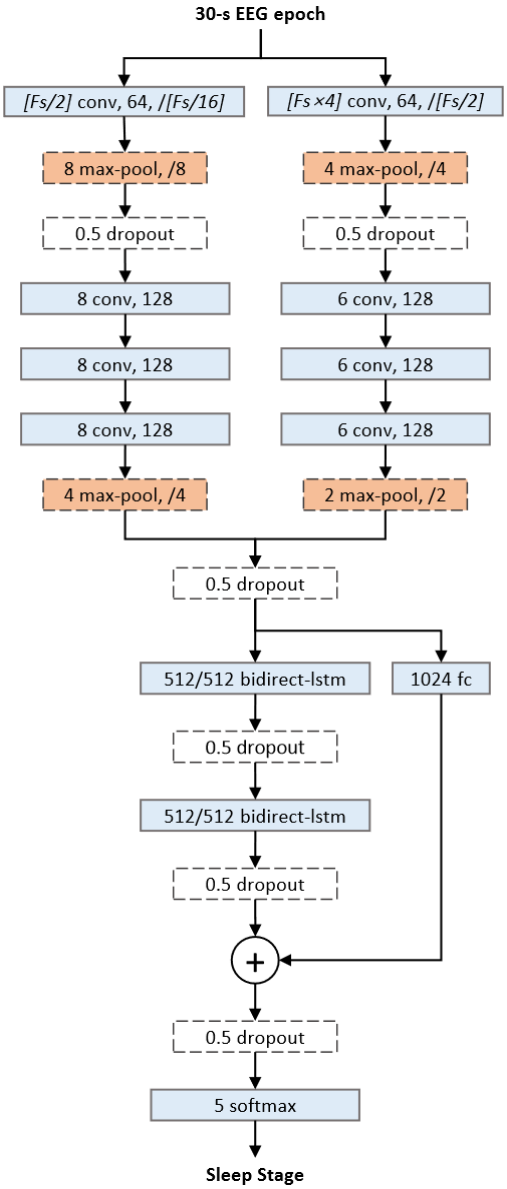


FIGURE 2.16 – Architecture du DeepSleepNet [165]. Fs est la fréquence d'échantillonnage du signal d'entrée, et indique la taille de filtre des couches de convolutions (*conv*) initiales. *max-pool* indique des couches de MaxPooling, et *bidirect-lstm* des couches de LSTM bidirectionnelles, avec une dimension de 512 pour l'espace latent. Les couches *dropout* sont précédées de leur probabilité.

bustSleepNet, dont l'architecture est illustrée figure 2.17. Les signaux des fenêtres $n-1$, n et $n+1$ sont représentés dans l'espace fréquentiel via FFT. Leurs caractéristiques sont encodées via une couche dense, puis les canaux sont combinés via attention si l'entrée est multidimensionnelle. Une suite de couches récurrente, dense puis d'attention termine l'encodage de caractéristiques. Les représentations des 3 fenêtres sont combinées dans des couches récurrentes, avant de faire la prédiction de l'état du sommeil.

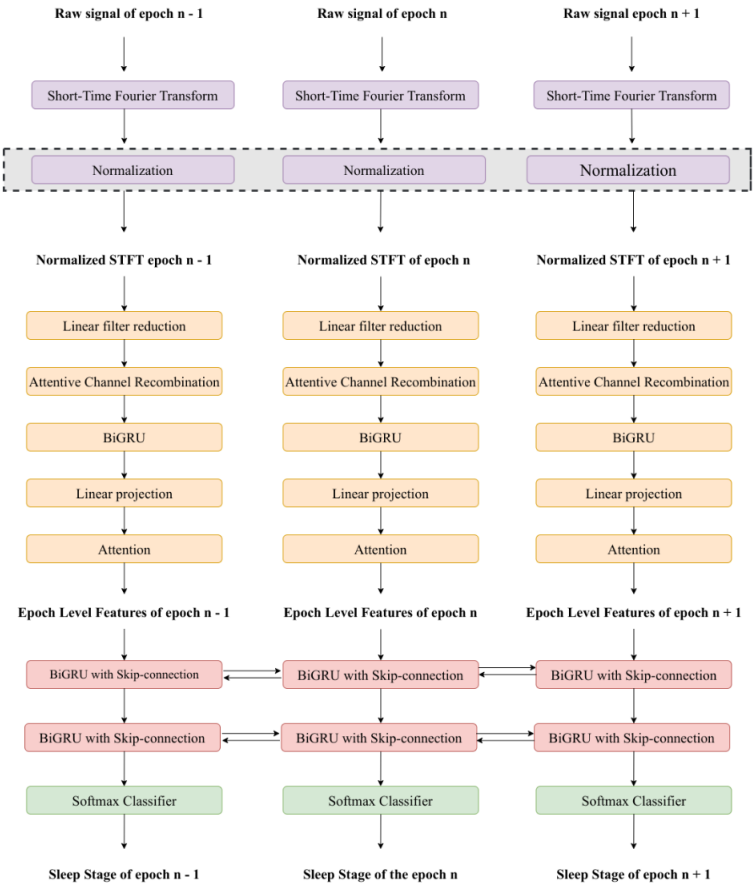


FIGURE 2.17 – Architecture du RobustSleepNet [66]

L'encodeur temporel Ft peut aussi être représenté avec un module d'attention (AttentionSleep [54]), ou par un Transformer, comme dans le SleepTransformer [131], ou SleepyCo [93]. L'architecture du SleepTransformer est illustrée figure 2.18 [9]. On obtient la représentation fréquentielle des L fenêtres $x^{(1:L)}$ via FFT puis on encode via un Transformer et de l'attention pour obtenir les caractéristiques des L fenêtres. On concatène ces représentations puis l'encodage temporel est réalisé par un autre Transformer. Enfin, on réalise la prédiction des L classes.

Enfin, et malgré le considérable essor des approches basées sur des réseaux de neurones, Sleep-Linear [174], basée sur des *hand-crafted features* et un classifieur linéaire,

9. Les modèles AttentionSleep et SleepyCo seront décrits dans le chapitre 5.

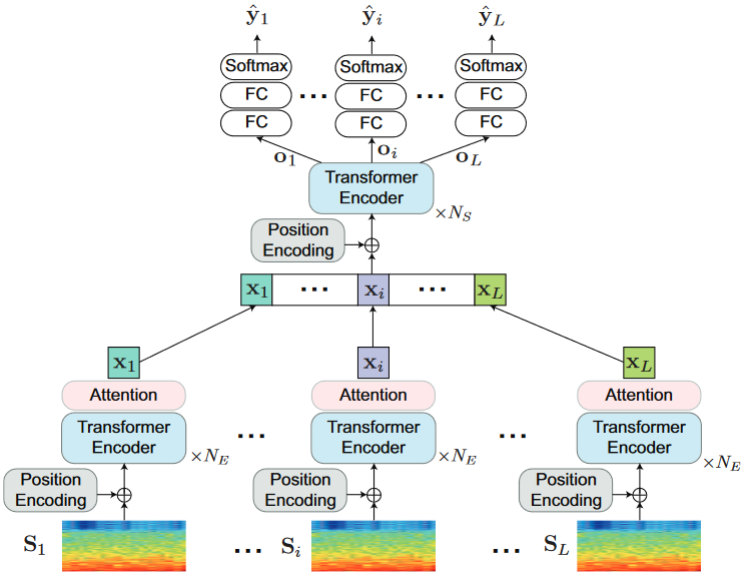


FIGURE 2.18 – Architecture du SleepTransformer [131]

obtient des résultats compétitifs. Nous détaillerons aussi cette méthode dans le chapitre 5.

Pour comparer ces différentes approches, nous adaptons les travaux de [126] qui présentent les scores κ de nombreux modèles sur les principaux jeux de données de sommeil. Nous remplaçons le score κ par le score macro F1 car il est plus souvent utilisé. Nous conservons uniquement les jeux de données SleepEDF-78, SHHS, MASS et DOD-O en raison du nombre significatif de résultats communiqués sur ces jeux. Enfin, nous choisissons les modèles pour illustrer les différentes architectures utilisées pour classifier le sommeil. Nous ajoutons des méthodes publiées après les travaux de [126], et présentons les résultats dans le tableau 2.3.

Les comparaisons entre les différentes approches sont assez délicates. En effet, tous les auteurs ne communiquent pas toujours les résultats de leurs modèles sur les différents jeux de données usuels, et n'utilisent pas toujours le score macro F1. Ils ne prennent pas non plus en compte le même nombre de fenêtres pour la prédiction, en particulier, utiliser des fenêtres futures empêche le fonctionnement temps réel, ce qui peut empêcher des applications comme la détection de fatigue des conducteurs en temps réel [110]. Néanmoins, l'approche la plus performante semble être SleepyCo [93], qui obtient le meilleur score sur les jeux de données SleepEDF-78, SHHS et MASS. Les résultats des différentes approches, pourtant basées sur diverses représentations des données et architecture des réseaux, restent similaires. Ceci est expliqué par [67], qui illustrent que les performances des modèles sont équivalentes, si ce n'est supérieures, à l'accord entre plusieurs annotateurs (sur le jeu de données DOD-O). Il est donc difficile d'envisager d'améliorer les performances des modèles sans changer les recommandations d'annotation.

Modèle	Année	Entrée	Encodeur de caractéristiques	Encodeur Temporel	Sleep EDF [83]	SHHS [186]	MASS [116]	DOD-O [67]
DeepSleepNet [165]	2017	Signaux bruts	CNN	RNN	0.71	N/A	0.82	0.81
SorsNet [159]	2018	Signaux bruts	CNN	CNN	0.73	0.76	N/A	N/A
SeqSleepNet [129]	2019	Temps-freq	RNN	RNN	0.76	N/A	0.79	0.78
IITNet [153]	2020	Signaux bruts	CNN-RNN	RNN	N/A	0.80	0.80	N/A
U-Sleep [125]	2021	Signaux bruts	U-Net	CNN	**0.79**	0.80	0.80	0.82
RobustSleepNet [66]	2021	Temps-freq	RNN+Attention	RNN	0.78	0.80	**0.83**	**0.83**
SleepTransformer [131]	2022	Temps-freq	Transformer	Transformer	**0.79**	0.80	N/A	N/A
SleepyCo [93]	2022	Signaux bruts	Transformer	Transformer	**0.79**	**0.81**	**0.83**	N/A
Sleep-Linear [174]	2023	Signaux bruts	Hand-crafted Features	Hand-crafted Features	**0.79**	N/A	0.82	N/A

TABLE 2.3 – Score macro F1 de différents modèles de classification de sommeil sur différents jeux de données. Extrait et adapté de [126], où nous avons remplacé le score κ par le score MF1.

2.3 Conclusion et discussion

Dans ce chapitre, nous avons présenté la classification de séries temporelles, en détaillant les deux principales approches pour résoudre la tâche : basée sur des distances et basée sur des caractéristiques. En utilisant les avantages de DTW, notamment sa capacité à comparer des séries de tailles différentes et son invariance aux décalages temporels, un simple classifieur 1-PPV obtient de bonnes performances.

Les approches basées sur des caractéristiques permettent d'utiliser les différentes architectures de réseau de neurones, comme les réseaux récurrents, convolutionnels ou à attention, pour représenter les séries temporelles de façon à mieux réaliser la tâche voulue. Cette représentation des séries est particulièrement importante pour les tâches autosupervisées où l'on compare l'entrée et la sortie du réseau. Ces tâches, comme la compression ou la détection d'anomalies, sont réalisées par des autoencodeurs dont nous avons présenté le fonctionnement. Les autoencodeurs nécessitent une fonction de coût de reconstruction pertinente pour bien représenter les signaux. Celle-ci est usuellement la MSE, malgré sa sensibilité au bruit et aux décalages temporels. Nous proposons donc de la remplacer par DTW. Cependant, le calcul de DTW est de complexité quadratique, et la métrique n'est pas différentiable, ce qui empêche de l'utiliser comme fonction de coût.

Ainsi, dans le chapitre 3, nous proposerons d'utiliser un autoencodeur basé sur un réseau siamois pour approximer DTW via une méthode différentiable. Dans notre approche, l'encodeur est conçu pour projeter les signaux de sorte que la distance euclidienne entre ces projections soit aussi proche que possible de la DTW entre les signaux originaux. Nous démontrerons que cette méthode d'approximation est non seulement fidèle à l'algorithme DTW original, mais possède également un temps de calcul peu sensible à la longueur des signaux.

Nous avons présenté un cas particulier de la classification de séries temporelles : la classification d'état du sommeil. Nous avons présenté les principaux jeux de données ainsi que la variété d'architectures de modèles utilisés pour résoudre cette tâche. Les performances des modèles sont similaires à l'accord entre les annotateurs donc peu d'améliorations sont envisageables sur ce point. Néanmoins, les données restent volumineuses : nous étudierons la possibilité de compresser les séries de sommeil en minimisant la perte de performance.

Pour ce faire, nous montrerons dans le chapitre 4 que nous pouvons apprendre un système de compression de bout en bout à l'aide d'un autoencodeur basé sur des couches TCN, ce qui permet de paramétrer le taux de compression. Nous évaluerons ce système de compression via le score de classification de sommeil sur les données compressées puis reconstruites pour simuler un cas réel d'application. Enfin, nous montrerons qu'utiliser notre approximation DTW basée sur une architecture siamoise comme fonction de coût permet d'obtenir de meilleures performances en classification sur les données reconstruites.

Enfin, dans le chapitre 5, nous étudierons l'impact du COVID long sur le sommeil. Pour cela, nous avons constitué un jeu de données en collaboration avec le centre du sommeil

VIFASOM. Ce jeu est constitué de polysomnographies de patients insomniaques après infection du COVID et est appelé COVISLEEP. Nous le mettons à disposition de la communauté. Nous comparerons les algorithmes de classification du sommeil présentés dans ce chapitre et montrerons la difficulté de cette tâche sur COVISLEEP, illustrant un intérêt du jeu de données pour la communauté. Enfin, nous étudierons la détection de COVID long et montrerons la difficulté de la tâche, notamment due à la forte variabilité entre les patients.

Chapitre 3

Approximation de DTW via un réseau de neurones convolutionnel

Sommaire

3.1 Introduction 63
3.1.1 Contexte et motivations 63
3.1.2 Définition du problème 64
3.2 État de l'art 65
3.2.1 Approximer pour accélérer DTW 65
3.2.2 Approximer pour rendre DTW différentiable 69
3.2.3 Approximation de fonction via réseaux de neurones 73
3.2.4 Approximation de la distance de Wasserstein via un réseau siamois 74
3.2.5 LDPS 75
3.2.6 Conclusion 77
3.3 Notre méthode pour approximer DTW 77
3.3.1 Architecture siamoise 77
3.3.2 Procédure d'apprentissage 79
3.3.3 Choix de l'encodeur 80
3.3.4 Choix du décodeur 81
3.3.5 Architecture directe 81
3.3.6 Conclusion 82
3.4 Expérimentations 82
3.4.1 Jeu de données 83
3.4.2 Implémentations et apprentissage 84
3.4.3 Vitesse 85
3.4.4 Fidélité 87
3.4.5 Différentiabilité 90
3.4.6 Conclusion 91

3.5 Application à d'autres jeux de données 91
3.5.1 Recherche de séries similaires 92
3.5.2 Classification 1-PPV 93
3.5.3 Classification de sommeil via prototypes 93
3.6 Synthèse des résultats 93
3.7 Conclusion du chapitre 96

Approximation de DTW via un réseau de neurones convolutionnel

3.1 Introduction

3.1.1 Contexte et motivations

Beaucoup d'applications nécessitent de bien comparer les séries temporelles, comme la vérification de signatures [122] ou la recherche de séries similaires [52]. Cela implique de choisir une métrique, fonction chargée de comparer les séries. En particulier, pour compresser des signaux temporels, il faut comparer les signaux reconstruits aux signaux originaux. Cette fonction d'erreur de reconstruction permet d'apprendre les poids des modèles compressant et décompressant les séries temporelles.

Les travaux de [52], [180] et [96] indiquent qu'en général, DTW (ou ses variations) est le meilleur choix pour comparer des séries temporelles, et donc une excellente fonction d'erreur de reconstruction. L'algorithme est cependant de complexité quadratique selon le nombre d'échantillons des séries comparées, et n'est pas différentiable, ce qui empêche de l'utiliser pour l'apprentissage de réseaux de neurones, basé sur la descente de gradient. Pour pallier à ces problèmes et afin d'utiliser DTW comme une fonction de coût, nous proposons de nous intéresser ici à l'approximation de DTW. Les réseaux de neurones, différentiables du fait de leur principe d'apprentissage, sont un moyen d'avoir une approximation dont la complexité est paramétrée par l'architecture du réseau.

L'objectif général d'apprentissage avec un réseau de neurones est d'approximer une fonction inconnue non linéaire permettant de résoudre une tâche donnée. Il est donc assez naturel de tenter d'utiliser des réseaux de neurones pour approximer des fonctions connues, en particulier pour des fonctions de complexité non linéaires. Ainsi, l'émergence des réseaux de neurones a permis l'introduction de diverses approximations d'équations de dynamique des fluides notamment, à l'image de [137] qui utilise un U-net (introduit par [139] pour la segmentation d'image médicale) pour générer les valeurs de champs de vitesse et de pression en deux dimensions. En particulier, la résolution des équations de Navier-Stokes, nécessaire à de nombreux domaines comme la modélisation des océans [104], demande de très importantes ressources en mémoire et en temps de calcul.

De manière analogue, dans ce chapitre, nous montrons comment approximer DTW comme une fonction en utilisant un réseau de neurones profond. Une telle approximation doit permettre de calculer le résultat de l'algorithme rapidement et de manière différentiable par rapport à ses entrées avec une erreur aussi basse que possible.

Nous posons ce problème d'approximation de DTW, puis présentons les principales approximations de DTW proposées dans la communauté : FastDTW [150], qui approxime DTW avec une complexité linéaire en temps, ucrDTW[134] qui utilise des bornes inférieures pour accélérer le calcul, softDTW[41] qui remplace l'opérateur min

pour rendre DTW différentiable, LDPS[100] qui transforme les signaux via des *shapelets* de sorte que la distance euclidienne des projections soit égale à la DTW des signaux et DTWNet[29] qui utilise le résultat du chemin d'alignement pour apprendre via DTW.

Cela ne permet cependant pas d'obtenir une approximation rapide et différentiable de DTW, nous nous intéressons donc ensuite à des méthodes d'approximation de fonction avec des réseaux de neurones, avec le cas particulier de l'approximation de la distance de Wasserstein. En effet, nous adaptons l'architecture utilisée par [40] pour approximer cette distance, car le principe est analogue à notre problème : approximer via un réseau de neurones le résultat d'une métrique comparant deux éléments. Nous présentons ensuite notre méthode : nous proposons deux architectures pour approximer DTW : une basée sur un autoencodeur siamois, adaptée de [40] et [100], qui projette les séries à comparer de sorte que la distance euclidienne des séries projetées soit égale à la DTW des séries initiales et une architecture régressant directement la valeur à prédire. Nous évaluons nos méthodes sur diverses tâches de comparaison de signaux : recherche du signal le plus proche, classification via 1-PPV, et classification par prototypes. Nous comparons les résultats de nos méthodes avec ceux des principales approximations de DTW. Enfin, pour étudier la capacité de généralisation de nos méthodes, nous montrons comment elles approximent DTW sur des données sur lesquelles elles n'ont pas appris.

3.1.2 Définition du problème

La tâche d'approximation de DTW est définie comme suit. Soient $x \in \mathbb{R}^{M \times K}$ et $x' \in \mathbb{R}^{N \times K}$ deux séries temporelles de longueurs respectives M et N et de dimension K . On définit comme vérité terrain $y = DTW(x, x')$ la valeur objective à approximer. L'objectif est donc d'apprendre une fonction

$$f : \left| \begin{array}{rcl} \mathbb{R}^{M \times K}, \mathbb{R}^{N \times K} & \longrightarrow & \mathbb{R} \\ x \ , \ x' & \longmapsto & \hat{y} \end{array} \right. \tag{3.1}$$

qui minimise la fonction de coût $L_f(y, \hat{y})$.

Dans ce chapitre, nous contraignons la tâche de la manière suivante :

— Les séries temporelles sont de longueurs possiblement variables $N, M \in [1, 3000]$ pour imiter la très importante capacité de DTW à comparer des séries de longueur différentes. Cela implique que les modèles utilisés doivent être capables de représenter des séries de tailles variables dans cet intervalle, sans nécessiter d'adapter leur architecture ou de réapprendre leurs poids.

— Les séries sont de dimension $K = 1$, caractéristique classique du traitement du sommeil. Ce choix permet à notre approximation de DTW de généraliser à d'autres jeux de données en sélectionnant un seul canal, alors que le choix de plusieurs dimensions impliquerait de modifier et donc de réapprendre les premières couches du modèle si le nombre de dimensions du nouveau jeu n'était pas égal à celui du jeu initial.

3.2 État de l'art

Dans cette section, nous présentons les principales approximations de DTW, en séparant celles dont le but principal est d'accélérer DTW de celles dont l'objectif est de rendre la distance différentiable. Nous étudions leurs avantages et inconvénients, et expliquons notre volonté d'approximer DTW avec un réseau de neurones. Nous présentons donc ensuite l'état de l'art de l'approximation de fonctions par des réseaux de neurones, en nous intéressant en particulier au domaine de la modélisation de dynamique des fluides, domaine d'application principal de cette technique. Puisque nous étudions le cas particulier de l'approximation d'une métrique, nous décrivons l'approche de [40] pour approximer la distance de Wasserstein, et justifions le choix d'un paradigme similaire pour approximer DTW.

Les séries temporelles pouvant être longues (par exemple, les enregistrements de sommeil peuvent comporter plusieurs centaines de milliers d'échantillons), il est courant d'extraire des fenêtres de signal, qui sont des morceaux de la série initiale. On note $x^{(i)}$ le i-ème morceau de la série temporelle x.

3.2.1 Approximer pour accélérer DTW

La complexité quadratique en temps et en espace de DTW freine son utilisation dans des cas d'applications nécessitant des comparaisons de longues séries temporelles (comme métrique d'un 1-PPV par exemple).

Les approches utilisées pour accélérer DTW se répartissent en trois catégories :

— Contraintes : On cherche à restreindre le nombre de coefficients calculés dans la matrice de coûts, c'est-à-dire ne pas comparer tous les échantillons des 2 séries temporelles. On réduit ainsi la complexité en temps de DTW.

— Abstraction : On cherche à appliquer la DTW sur une représentation plus compacte des données. En fonction de ce qui est réduit (la dimension temporelle ou le nombre de canaux des séries), on réduit la complexité de DTW.

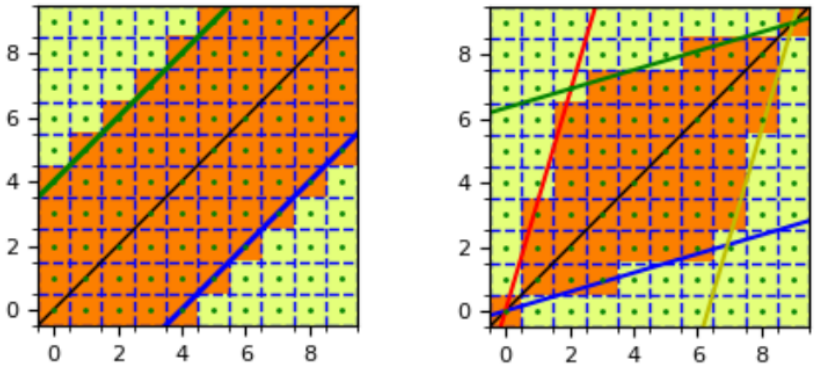
— Indexation : On cherche à utiliser des bornes inférieures pour diminuer le nombre d'exécutions de DTW lors de la classification ou du regroupement des séries temporelles. On n'adresse pas directement la complexité de l'algorithme, mais on cherche à moins l'utiliser pour accélérer la tâche finale.

Nous détaillons des exemples d'utilisation de ces approches dans la suite de cette section.

Les deux contraintes les plus fréquentes sont l'utilisation de la bande de Sakoe-Chiba [146] et du parallélogramme d'Itakura [77]. La bande de Sakoe-Chiba restreint la recherche du chemin d'alignement dans une bande de rayon r autour de la diagonale de la matrice. Ainsi, on ajoute une contrainte à l'équation 2.4 : $j - r \leq i \leq j + r$.

Le parallélogramme d'Itakura définit une pente autour de la diagonale, résultant en la forme de parallélogramme de la zone considérée pour le chemin d'alignement. On

1. https://pyts.readthedocs.io/en/stable/



(a) Bande de Sakoe-Chiba (b) Parallélogramme d'Itakura

FIGURE 3.1 – Restrictions de l'espace de recherche du chemin d'alignement dans la matrice de coût [1]

illustre le fonctionnement des deux méthodes dans la figure 3.1. Naturellement, puisque ces bandes restreignent la position du chemin d'alignement retourné, elles peuvent introduire des erreurs si le meilleur chemin ne se trouve pas dans la zone en question.

Ainsi, ces contraintes fonctionnent bien dans les cas où les séries temporelles ont une faible variance dans leur alignement temporel et si le chemin d'alignement est proche de la diagonale, c'est-à-dire que la valeur de DTW est proche de la distance euclidienne. Cependant, les contraintes fonctionnent mal si les tâches nécessitent de détecter des courts événements se trouvant à différents instants en fonction des séries : par exemple la recherche d'une anomalie de battement de cœur dans des ECG [166]. Dans ce cas, le chemin d'optimisation peut s'éloigner considérablement d'une transformation linéaire et presque toute la matrice de coûts doit être évaluée pour trouver un chemin d'alignement optimal. L'abstraction crée de l'erreur, en particulier pour des séries non stationnaires, irrégulières ou très longues, puisqu'il est difficile d'étendre les résultats obtenus à faible échelle. Enfin, l'indexation est très utile pour la recherche de séries similaires, mais n'est pas applicable si on veut une mesure précise de similarité entre deux séries, comme dans le cas d'une fonction de coût de reconstruction.

Beaucoup d'algorithmes existent pour accélérer DTW, en utilisant une combinaison des principes mentionnés précédemment. Les 2 méthodes les plus utilisées dans la littérature, FastDTW [150] et ucrDTW [134], sont présentées dans la suite.

3.2.1.1 FastDTW

FastDTW, introduit par [150], est une approximation de DTW de complexité linéaire en temps et en espace. Elle combine contraintes et abstractions et comprend trois opérations principales :

— le ***coarsening*** : on réduit la taille de la série temporelle en prenant la moyenne entre 2 points adjacents (équivalent à *l'average pooling* en 1D avec une fenêtre de taille 2). Cette opération est réalisée à plusieurs reprises pour obtenir des représentations de la série temporelle à plusieurs échelles. C'est une technique

d'abstraction.

— La **projection** calcule le chemin d'alignement entre les 2 séries à une échelle donnée et en déduit le chemin projeté à l'échelle supérieure, c'est-à-dire avec un degré de *coarsening* plus faible.

— Le ***refinement*** trouve un chemin d'alignement optimal autour du chemin extrait de la projection. La zone de recherche est définie par un paramètre r, à l'instar de la bande de Sakoe-Chiba.

La complexité de DTW est en $\mathcal{O}(N * M * K)$, avec N, M les tailles des séries temporelles et K le nombre de canaux (ou dimensions) des séries temporelles. En effet, tous les composants de la matrice de coût doivent être remplis pour obtenir le chemin d'alignement.

L'avantage de FastDTW est alors qu'une fois les trois opérations mentionnées précédemment terminées, la complexité de FastDTW est en $\mathcal{O}(N*2r)$ si $N > M$ car il suffit de remplir les composants dans un rayon r autour du chemin d'échelle précédente. r peut être vu comme un compromis entre fidélité et complexité : si $r \geq M/2$ alors tous les coefficients de la matrice sont évalués et FastDTW se comporte comme DTW. À l'inverse, quand r tend vers 0, FastDTW est plus rapide que DTW mais risque de ne pas trouver le chemin optimal et donc perdre en fidélité. Le choix usuel est $r = 0.05*N$. On notera pour simplifier $r = 0.05$.

Les performances de l'algorithme, bien que largement utilisé dans la littérature, ont été remises en cause par [181]. Les auteurs séparent les cas d'applications de comparaisons de séries temporelles selon la longueur des séries temporelles considérées (N) et selon le taux d'alignement nécessaire (W, exprimé en pourcentage de N, représente l'espace de recherche du chemin d'alignement optimal et est déterminé expérimentalement par *grid search* selon un score de classification). Selon leur analyse, FastDTW est plus rapide que DTW contraint ($cDTW_W$) uniquement lorsque les séries sont longues ($N > 1000$) et que W est très grand ($W \approx 100$). Néanmoins, les séries de sommeil étant usuellement de taille 3000, en présumant $W \approx 100$, nous nous situons dans le cas où FastDTW est plus rapide que $cDTW_W$ et, par conséquent, nous utiliserons FastDTW pour les comparaisons dans nos expérimentations futures.

On remarque que FastDTW applique l'algorithme DTW à des séries temporelles réduites, et souffre donc du même défaut : la non-différentiabilité.

3.2.1.2 ucrDTW

ucrDTW, introduit par [134], utilise une série d'optimisations, principalement de type indexation, pour accélérer le calcul d'exploration de données (*data mining)* dans de très grands volumes de séries temporelles. Pour la suite, on considère la tâche de trouver la série temporelle candidate $\hat{C}$ la plus proche d'une série *query* $Q \in \mathbb{R}^{N \times K}$ dans un jeu de données D, avec $C_c \in \mathbb{R}^{M,K}$ la série candidate à comparer. On note DTW_{min} la valeur la plus basse de DTW obtenue avant d'avoir traité la série C_c. Ces optimisations sont notamment :

1) Ignorer la racine carrée dans le calcul de la distance euclidienne.

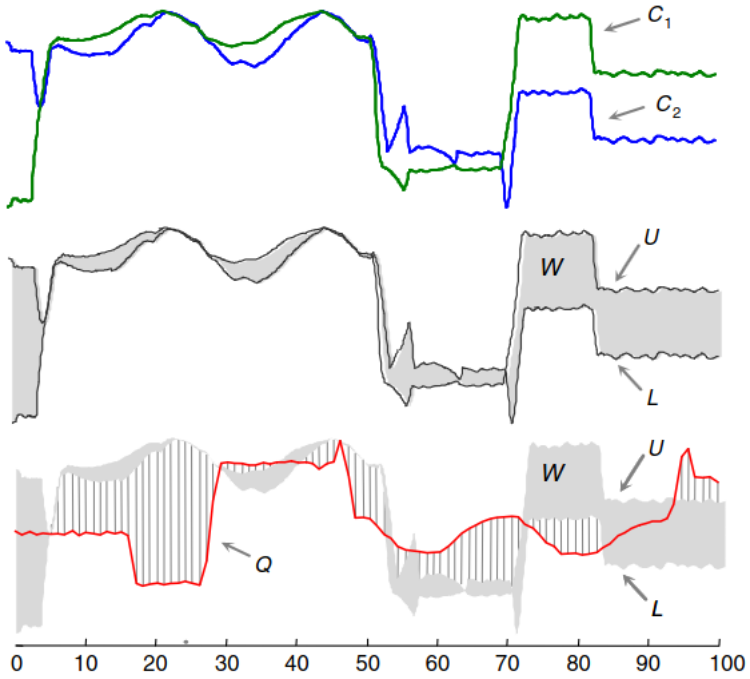


FIGURE 3.2 – Borne inférieure de Keogh. En haut sont les séries à comparer, au milieu le *Wegde* obtenu, et en bas la borne de Keogh. Extrait de [84].

2) Utiliser des bornes inférieures. Diverses bornes inférieures de DTW existent dans la littérature, mais ucrDTW utilise principalement les bornes suivantes :

— $LB_{Kim}FL$[87] qui, modifiée par [134], est définie par :

$$LB_{Kim}FL = ED(C_c[0], Q[0]) + ED(C_c[M], Q[N]) \tag{3.2}$$

avec ED la distance euclidienne. Avec $K = 1$, la borne est de complexité $\mathcal{O}(1)$.

— LB_{Keogh} [84]. On la définit en choisissant k séries dans D. On introduit ensuite deux séries temporelles :

$$U \text{ tel que } U_i = \max(D_i^{(1)}, ..., D_i^{(k)}) \tag{3.3}$$

et

$$L \text{ tel que } L_i = \min(D_i^{(1)}, ..., D_i^{(k)}) \tag{3.4}$$

. On appelle $Wedge$ l'ensemble $\{U, L\}$. On a alors :

$$LB_{Keogh}(Q, Wedge) = \sqrt{\sum_{i=1}^{N} \begin{cases} (q_i - U_i)^2 & \text{if } q_i > U_i \\ (q_i - L_i)^2 & \text{if } q_i < L_i \\ 0 & \text{otherwise} \end{cases}} \tag{3.5}$$

Cette borne est de complexité $\mathcal{O}(N)$. On l'illustre dans la figure 3.2.

D'autres bornes, comme celles proposées par [147], [52] et [187] peuvent aussi être utilisées.

Dans tous les cas, on obtient $b = borne(Q, C_c)$ et si $b > DTW_{min}$, alors on ignore la candidate C_c. On range ainsi les bornes $b^{(i)}$ selon l'ordre croissant de leur complexité, puis on calcule chaque borne $b^{(i)}$ jusqu'à ce que l'on trouve $b^{(i)} > DTW_{min}$ pour exclure la série candidate C_c.

3) Utiliser des bornes inférieures partielles. Si le calcul de borne inférieure ne permet pas d'éliminer C_c, on calcule la DTW entre Q et C_c jusqu'à l'échantillon $i \in [1, N[$ et on y ajoute le calcul de la borne de l'échantillon $i+1$ à N. Si $\exists i \in [1, N[$ tel que :

$$DTW(Q[1:i], C_c[1:i]) + borne(Q[i+1:N], C_c[i+1:N]) > DTW_{min} \tag{3.6}$$

alors on ignore la candidate C_c.

4) Réordonner l'ordre des bornes inférieures partielles. Au lieu de commencer à $i = 1$, on recherche les points contribuant le plus à la valeur finale de DTW. Puisque les séries sont Z-normalisées, cela correspond simplement aux échantillons de plus hautes valeurs. Ainsi, on calcule la DTW avec les points ayant le plus de chance d'impacter la valeur finale de l'algorithme en premier, permettant de plus rapidement éliminer C_c.

5) Calculer la normalisation des séries temporelles *online* et donc l'arrêter si une des conditions précédentes n'est pas satisfaite.

Enfin, ucrDTW utilise la bande de Sakoe-Chiba pour restreindre le calcul de DTW. Les techniques de borne inférieure peuvent être utilisées même pour le calcul d'une seule comparaison de série temporelle, accélérant significativement le calcul (au prix d'une relative inexactitude des calculs, notamment avec la bande $r = 0.05$, choisie "par défaut" par les auteurs.)

Nous avons présenté les principales méthodes existantes pour réduire la complexité de l'algorithme DTW : les méthodes FastDTW et ucrDTW, qui regroupent à notre connaissance toutes les méthodes proposées par la communauté. Les récents travaux de [181] ont cependant montré que FastDTW n'était plus rapide que pour de longues séries dans lesquelles on ne peut pas réduire l'espace de recherche du chemin a priori. L'utilisation de bornes inférieures dans la méthode ucrDTW permet d'accélérer le calcul de DTW dans tous les cas, au prix d'une fidélité moindre dans les applications autres que la recherche de séries similaires : nous illustrerons dans la suite les performances des différentes méthodes en classification. Cependant, tout comme FastDTW, ucrDTW reste non différentiable.

3.2.2 Approximer pour rendre DTW différentiable

La lenteur de l'algorithme DTW, particulièrement sur de longues séries, est un frein à son utilisation en tant que fonction d'erreur pour un apprentissage bout-à-bout. Le principal problème est cependant la non-différentiabilité de la métrique. Ainsi, plusieurs approximations existent pour contourner le problème : softDTW [41] qui remplace l'opérateur min par une version différentiable, et DTWNet [29], qui dérive uniquement la forme obtenue via le chemin d'alignement. Ces méthodes sont détaillées dans la suite de cette section.

3.2.2.1 Non-différentiabilité de DTW

Dans cette section, nous discutons de la non-différentiabilité de DTW, ce qui permet de comprendre comment certaines approximations rendent DTW différentiable.

Pour rendre DTW différentiable, il est nécessaire de comprendre pourquoi la métrique ne l'est pas : c'est dû à l'opérateur min. On reprend le raisonnement développé dans [168], et on introduit le théorème suivant :

Theorem 1. *Soit Ψ un espace métrique, X un espace normé, et Π un sous-ensemble de Ψ. Soit v la fonction de valeur optimale telle que : $v(x) = \inf_{\pi \in \Pi} f(x; \pi)$*

On suppose que :
1) $\forall \pi \in \Psi, \rightarrow f(x; \pi)$ est différentiable
2) $f(x; \pi)$ et sa dérivée $D_x f(x; \pi)$ sont continues sur $X \times \Psi$.
Si, pour $x^0 \in X$, $\pi \rightarrow f(x^0, \pi)$ admet un unique minimiseur π^0 sur Π alors v est différentiable en x^0

En utilisant le théorème 1 introduit par [24], on prouve qu'en choisissant :

$$v(x) = DTW(x, x_{ref}) = \min_{\pi \in A(x, x_{ref})} \langle A_\pi, d_{ED}(x, x_{ref}) \rangle^{\frac{1}{2}} \tag{3.7}$$

alors v est différentiable sauf si :

- $DTW(x, x_{ref}) = 0$ car la racine carrée n'est pas différentiable et donc ne satisfait pas la condition 1
- il existe plusieurs chemins optimaux et donc pas d'unicité du minimiseur π^0

On prouve ainsi que l'on ne peut pas garantir la différentiabilité de DTW, et donc que l'on ne peut théoriquement pas utiliser DTW comme fonction de coût, à cause de l'opérateur min.

3.2.2.2 softDTW

Pour contourner cette limitation, [41] définit l'opérateur *soft-min* par :

$$min^\gamma(x_1, ..., x_N) = \begin{cases} \min\limits_{i \leq N} x_i & \text{if } \gamma = 0 \\ -\gamma \, log \sum_{i=1}^{N} e^{-x_i/\gamma} & \text{if } \gamma > 0 \end{cases} \tag{3.8}$$

où γ est un hyperparamètre lissant le *soft-min*. Ainsi, on contourne l'élément qui rendait l'algorithme non différentiable.

On réinjecte ensuite la définition du *soft-min* dans l'équation 2.4 pour obtenir la *softDTW* :

$$softDTW^\gamma(x, x') = \min_{\pi \in A(x, x')}^{\gamma} \sum_{(i,j) \in \pi} d_{ED}(x_i, x'_j)^2 \tag{3.9}$$

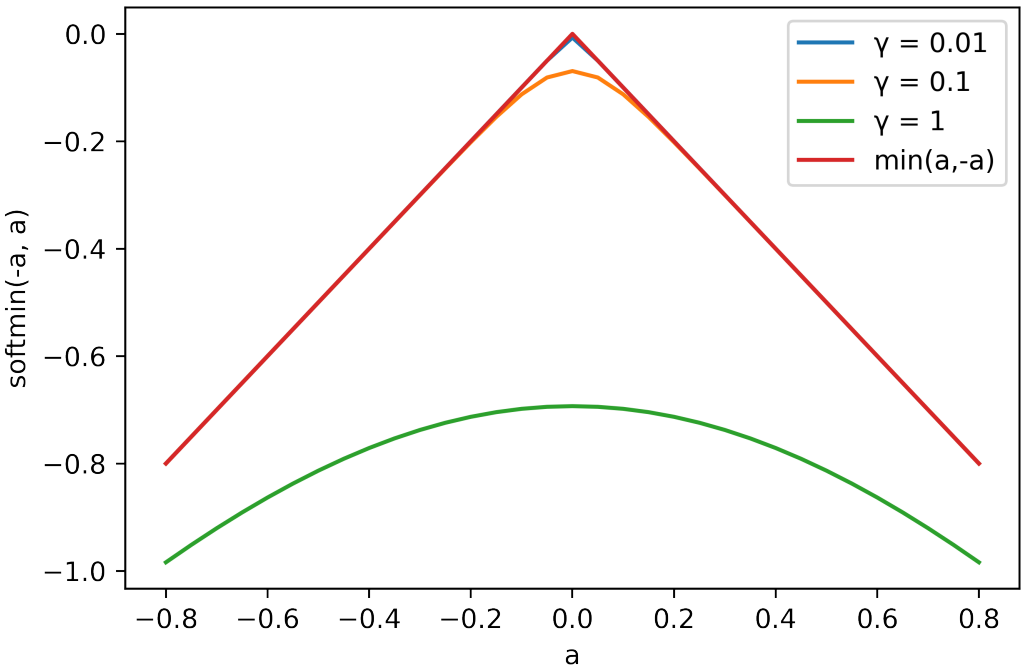


FIGURE 3.3 – Résultat de l'opérateur $softmin$ pour différentes valeurs de γ, en comparaison avec l'opérateur min. Inspiré par [168].

Naturellement, ce changement impacte le résultat de l'algorithme. Pour l'étudier, on illustre figure 3.3 le comportement de *soft-min* par rapport à min en fonction de γ. On voit que quand γ tend vers 0, *soft-min* tend vers min. On a donc :

$$softDTW^{\gamma}(x, x') \xrightarrow{\gamma \to 0^+} DTW(x, x')^2 \tag{3.10}$$

Si elle est différentiable, softDTW présente d'autres freins à son utilisation en tant que fonction de coût. Le changement d'opérateur n'impacte pas la complexité de l'algorithme, qui reste donc quadratique. L'opération de rétro-propagation est de fait aussi de complexité quadratique. De plus, comme indiqué dans l'équation 3.9, contrairement à la méthode DTW, qui ne considère que le meilleur alignement, softDTW prend en compte tous les alignements possibles, le résultat étant la somme pondérée par la qualité du chemin de tous les chemins possibles. Ainsi, à la différence de DTW, en général :

$$softDTW(x, x) \neq 0 \tag{3.11}$$

L'invariance au décalage temporel de DTW est perdue pour la même raison.

Pour adresser ces difficultés et toujours obtenir un résultat positif ou nul de la métrique, [22] introduit la divergence softDTW par :

$$div^{\gamma}_{softDTW}(x, x') = softDTW^{\gamma}(x, x') - \frac{1}{2}(softDTW^{\gamma}(x, x) + softDTW^{\gamma}(x', x')) \tag{3.12}$$

Ainsi, on retrouve $div^{\gamma}_{softDTW}(x, x) = 0$. La divergence *softDTW* est donc plus adaptée pour des tâches de reconstruction de signaux, où l'objectif du modèle est de retrouver le signal original. L'inconvénient évident est que cette méthode nécessite de calculer la *softDTW* trois fois, aggravant la lenteur des calculs.

Nous avons présenté *softDTW*, qui rend DTW différentiable en introduisant le *softmin*. Dans la suite, nous présentons une approche exploitant une autre propriété de DTW pour obtenir une approximation différentiable, nommé DTWNet [29] : lorsque le calcul du chemin d'alignement est obtenu, le calcul de DTW revient à la somme des distances entre les points alignés, qui est différentiable.

3.2.2.3 DTWNet

Comme montré dans l'équation 2.5, une fois que le chemin d'alignement est déterminé, le calcul de DTW est simplement la somme des distances entre les points alignés :

$$\mathrm{DTW}(x, x') = \sqrt{\|x'_0 - x_0\|_2^2 + \|x'_1 - x_0\|_2^2 + \|x'_2 - x_1\|_2^2 + \dots} \tag{3.13}$$

[29] exploitent cette propriété pour utiliser DTW comme noyau afin d'extraire des caractéristiques des signaux, à l'image d'un filtre de convolution : $z = DTW(x, w)$ avec x la série en entrée et w les poids de la couche du réseau. Ils créent ainsi des couches de réseaux de neurones, composées de filtres DTW et de couches linéaires. Cela nécessite cependant que le filtre DTW soit différentiable pour pouvoir *backpropager* le gradient.

Pour ce faire, ils obtiennent le chemin d'alignement via l'algorithme DTW classique, et appliquent la différentiation uniquement sur le chemin obtenu. Cette méthode permet d'éviter une *backpropagation* de complexité quadratique. En effet, puisque nous fixons $K = 1$, le calcul de $\|\hat{x}_0 - x_0\|_2^2$ est de complexité $\mathcal{O}(1)$. Alors, puisque le chemin compte au maximum $N + M$ éléments, la complexité de *backpropagation* est en $\mathcal{O}(N + M)$. Le calcul du chemin d'alignement nécessite néanmoins toujours $\mathcal{O}(N * M)$ opérations, et n'est pas parallélisable.

En résumé, DTW n'est pas différentiable à cause de l'opérateur min. Pour utiliser la métrique comme fonction de coût, [41] introduit l'opérateur *soft-min*. Cela ajoute cependant un hyperparamètre γ qu'il faut déterminer par *grid search* pour chaque application. Pour éviter cette difficulté, [29] choisit de premièrement calculer le chemin d'alignement de DTW entre deux séries et de simplement dériver la forme ainsi obtenue. Cependant, les deux approches conservent la complexité quadratique de DTW, et ne produisent pas de résultats exacts. À notre connaissance, il n'existe pas d'autre approche permettant de dériver DTW directement.

Pour obtenir une approximation de DTW différentiable et rapide, il faut donc s'affranchir de l'algorithme DTW et simplement approximer la valeur de similarité. Cela revient à voir DTW comme une fonction boîte noire dont on cherche à imiter les résultats : c'est alors un problème d'approximation de fonction. Les réseaux de neurones

étant différentiables par défaut [2] et capables d'approximer toute fonction continue [76], nous présentons dans la suite l'approximation de fonction avec des réseaux de neurones, en particulier dans le domaine de la dynamique des fluides.

3.2.3 Approximation de fonction via réseaux de neurones

La résolution numérique d'équations différentielles partielles (PDEs), cruciales dans des domaines comme la gestion de portefeuille en finance [120] ou la physique quantique [28], est rendue quasi impossible par le fléau de la dimension (introduit par [18]). Ainsi, l'utilisation de réseaux de neurones pour approximer les solutions numériques s'est grandement popularisée dans la communauté ces dernières années. En effet, leur capacité à approximer un grand nombre d'équations différentielles en fait un excellent outil de prototypages quand les légères erreurs d'approximation commises sont acceptables [21].

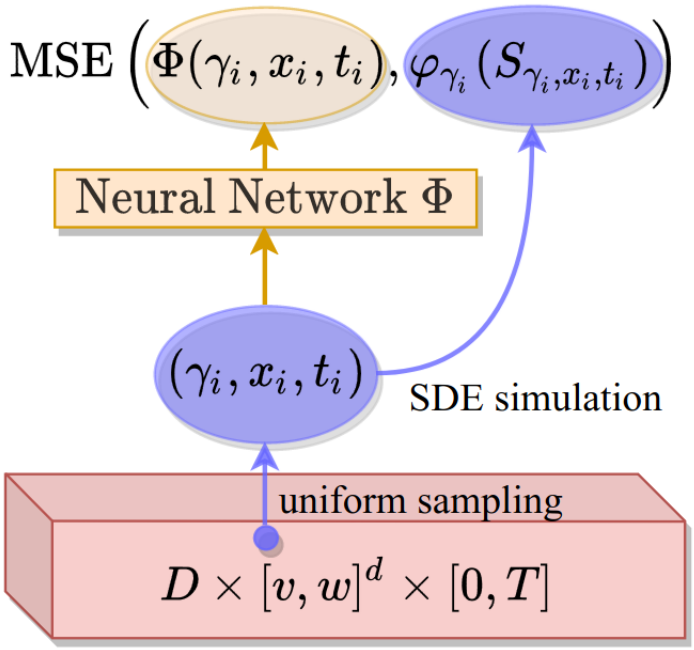


FIGURE 3.4 – Exemple de paradigme d'approximation d'équation différentielle partielle de Kolmogorov. Extrait de [20]. Les variables d'entrée x, t et γ sont échantillonnées de $X \sim \mathcal{D} \times [\nu, w]^d \times [0, T]$, où $\mathcal{D} \times [\nu, w]^d \times [0, T]$ est le domaine spatio-temporel d'intérêt de l'équation. On nomme les conditions initiales de l'équation φ_γ . La valeur cible $y = \varphi_\gamma(S_{\gamma,x,t}) \in \mathbb{R}$ est obtenue explicitement ou via des solveurs numériques (S) selon les types d'équations à résoudre.

Un exemple de ce paradigme, dans le cas où la valeur cible y est obtenue via simulation, est donné figure 3.4. L'objectif est de minimiser la MSE entre les valeurs prédites par le modèle Φ et la valeur réelle $y = \varphi_\gamma(S_{\gamma,x,t}) \in \mathbb{R}$:

$$\min_{\Phi} MSE(y, \Phi(\gamma, x, t)) \tag{3.14}$$

2. nous l'avons rappelé dans la section 2.1.3.2

Ainsi, on peut par exemple approximer l'équation de chaleur en d dimensions définie par :

$$\begin{aligned} \partial_t u(t,x) &= \Delta u(t,x) \quad (t,x) \in [0,T] \times \mathbb{R}^d \\ u(0,x) &= \|x\|^2 \qquad\qquad\qquad\quad x \in \mathbb{R}^d, \end{aligned} \tag{3.15}$$

On pré-calcule d'abord les valeurs cibles $y = \|x\|^2 + 2 * td$ puis on apprend les poids du réseau de neurones selon la fonction de coût $\mathcal{L} = MSE(\Phi(x,t), y)$.

Il y a peu de contraintes concernant l'architecture du réseau de neurones utilisé. [17] crée un réseau basé sur la suite de couches suivante :
Entrée→BN→(Dense→BN→TanH)→(Dense→BN→TanH)→Dense→BN→Sortie. BN représente les couches de *batch normalization*. En utilisant le même modèle, [21] obtiennent une erreur MSE de l'ordre de $10e^{-4}$.

Bien que ces équations soient dépendantes du temps, les réseaux n'utilisent pas d'architectures adaptées comme des RNN, convolutions dilatées ou attention. En effet, les valeurs de la fonction à approximer sont générées séquentiellement à partir de l'état initial et des états précédemment générés.

Ces méthodes permettent d'approximer des fonctions avec une bonne précision. Néanmoins, DTW nécessite de comparer les 2 séries temporelles x, x' en même temps pour trouver le chemin d'alignement. Ainsi, nous avons besoin d'une approche qui prenne en compte les 2 signaux d'entrée dans leur globalité. Plus généralement, cela revient à obtenir $\hat{y} = \phi(x, x')$ avec un réseau de neurones ϕ, et à minimiser $MSE(\hat{y}, y)$ avec $y = dist(x, x')$. Cette tâche est abordée par [40], qui cherchent à approximer la distance de Wasserstein. Nous présentons donc dans la suite leur méthode.

3.2.4 Approximation de la distance de Wasserstein via un réseau siamois

Avec [40], Courty et al. introduisent une façon d'approximer la distance de Wasserstein W_2^2, notamment utilisée pour comparer des distributions de probabilité empirique. L'objectif est, à partir d'une paire d'histogrammes $(x^{(i)}, x'^{(j)})_{(i,j)\in 1..N}$, de prédire $\hat{y}^{(i,j)}$ tel que $\hat{y}^{(i,j)} = y^{(i,j)} = W_2^2(x^{(i)}, x'^{(j)})_{(i,j)\in 1..N}$.

L'architecture de leur approche siamoise est illustrée figure 3.5. Les auteurs construisent un auto-encodeur basé sur un réseau siamois [36]. Les deux signaux d'entrée $x^{(i)}, x'^{(j)}$ sont envoyés séparément à un encodeur ϕ, qui les projette en deux représentations $z^{(i)} = \phi(x^{(i)})$ et $z'^{(j)} = \phi(x'^{(j)})$. L'objectif est que la distance euclidienne entre $z^{(i)}$ et $z'^{(j)}$ soit égale à la distance de Wasserstein entre les signaux initiaux, c'est-à-dire $\|z^{(i)} - z'^{(j)}| = W_2^2(x^{(i)}, x'^{(j)})$.

Pour faciliter l'apprentissage (comme illustré par [185]), un décodeur ψ (simplement composé de couches denses) tente de retrouver $x^{(i)}$ et $x'^{(i)}$ à partir de $z^{(i)}$ et $z'^{(i)}$, respectivement. L'encodeur et le décodeur sont appris conjointement via deux fonctions de coûts : la fonction MSE compare la distance euclidienne entre les signaux projetés et la valeur cible de Wasserstein tandis que la divergence de Kullback-Leibler (KL) [90] compare les signaux reconstruits par le décodeur aux signaux originaux. L'objectif

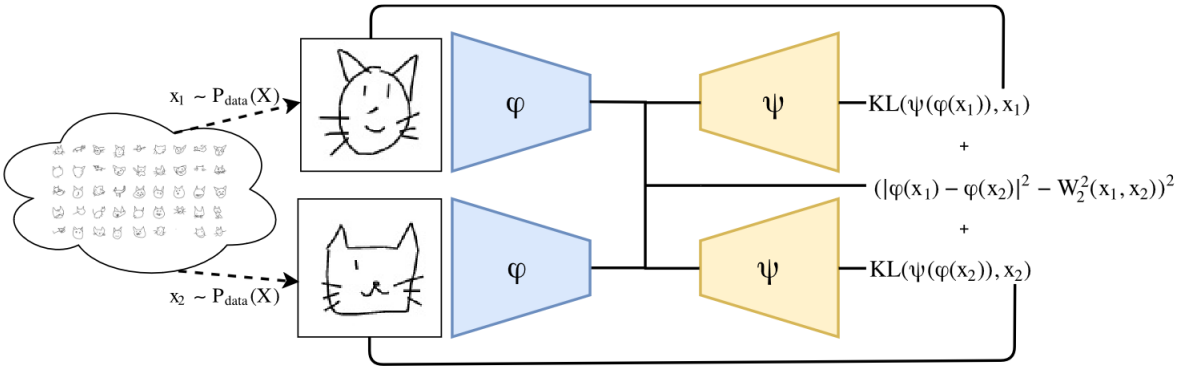


FIGURE 3.5 – Approximation de la distance de Wasserstein par [40]

de l'apprentissage est alors d'obtenir :

$$\min_{\phi,\psi} \sum_{i,j} \left| \|z^{(i)} - z'^{(j)}\|^2 - y^{(i,j)} \right|^2 + \lambda \left(KL(\psi(z^{(i)}) - x^{(i)}) + KL(\psi(z'^{(j)}) - x'^{(j)}) \right) \quad (3.16)$$

avec λ un réel servant à équilibrer les fonctions de coût.

Cette architecture permet une approximation rapide et fiable de la distance de Wasserstein.

L'approximation de métriques en comparant la distance euclidienne de signaux projetés est intéressante car elle permet d'obtenir un espace dans lequel les algorithmes de *clustering* comme le Plus Proche Voisin sont optimisés. Pour ces raisons, [100] utilise des *shapelets* pour transformer les signaux de sorte que la distance euclidienne entre les *shapelets* soit égale à la DTW originelle.

3.2.5 LDPS

Les *shapelets*, introduites par [182], sont des sous-séquences de séries temporelles permettant de mieux discriminer les séries. [71] utilise des *shapelets* pour générer des caractéristiques en calculant une distance entre les séries et un ensemble de *shapelets*. En se basant sur cette idée, [65] propose d'apprendre les *shapelets* au lieu de les extraire des séries temporelles. Le principe est d'utiliser une fonction de cout de classification pour apprendre les *shapelets* via descente de gradient. Ensuite, en inférence, la méthode de création de caractéristiques de [71] est utilisée, ce qui permet d'améliorer les performances de classification du modèle.

En se basant sur ces travaux, [100] propose la méthode *Learn DTW-Preserving Shapelets* (LDPS), permettant d'approximer DTW via un ensemble de *shapelets* (notées *sp*). L'objectif est que la distance euclidienne des *shapelets* soit la plus proche possible de la DTW entre les séries originelles, à l'image de [40] avec la distance de Wasserstein. Cependant, à la différence de [40], aucun réseau de neurones n'est utilisé. En effet, l'objectif n'est pas d'apprendre la transformation des séries en *shapelets*, mais d'apprendre

un ensemble S de K *shapelets* de longueur possiblement variable. On transforme x en la *shapelet* la plus corrélée à x parmi S.

L'apprentissage de ces *shapelets* se fait via descente de gradient, en optimisant la fonction de coût suivante :

$$\mathcal{L}(x, x') = \frac{1}{2}\left(DTW(x, x') - \beta \left\|z - z'\right\|_2\right)^2 \tag{3.17}$$

avec x, x' les deux séries temporelles, z, z' leurs représentations par *shapelets* respectives et β un réel servant d'équilibrage (qui est ici appris). Cette méthode d'apprentissage prouve que LDPS peut être vue comme une approximation différentiable de DTW.

Par rapport à la méthode siamoise d'approximation de distance de Wasserstein de [40], LDPS permet de s'affranchir de la recherche de l'architecture et des hyperparamètres de l'encodeur et du décodeur. Elle nécessite cependant de déterminer le nombre et la longueur des *shapelets* : plus celles-ci sont longues et nombreuses, plus l'approximation sera fidèle, mais aussi plus le calcul de la distance sera long. [100] choisissent de former 3 sous-groupes de *shapelets* de longueurs L_s respectivement égales à 15%, 30% et 45% de la longueur N des séries x. Chaque sous-groupe contient $N_{sp} = 10 * \log(N - L_s)$ éléments. En inférence, la complexité de LDPS pour chaque sous-groupe i est donc :

$$\mathcal{O}_i(\frac{\alpha_i}{100} * N * 10 * \log(N(1 - \frac{\alpha_i}{100}))) \tag{3.18}$$

avec $\alpha_i \in [15; 30; 45]$ le pourcentage de taille de N choisi. Cette complexité reste significative. Nous comparerons dans la suite de ce chapitre les temps d'exécution nécessaires à différentes approximations de DTW.

Récemment, [6] ont proposé l'approche *Constrained DTW-Preserving Shapelets* (CDPS), qui permet d'étendre LDPS à des cas de *clustering* sous contraintes. Puisque nous ne réalisons pas de *clustering* dans nos expérimentations, nous utiliserons LDPS dans nos comparaisons.

Nous proposons donc de nous baser sur l'approximation de distance de Wasserstein et LDPS pour approximer DTW via un réseau siamois, dans le but d'obtenir une approximation de DTW fiable, différentiable et peu impactée par la taille des séries.

Nous avons présenté diverses approximations de DTW permettant d'accélérer ou de rendre l'algorithme différentiable, ainsi que diverses méthodes d'approximation de fonction via des réseaux de neurones. Cependant, si la réduction de complexité et l'ajout de différentiabilité sont deux apports nécessaires pour une bonne approximation de DTW, le premier critère reste la fiabilité : une bonne approximation doit conserver les avantages de DTW et se comporter de manière similaire. Pour vérifier cette propriété, il convient d'évaluer les métriques avec diverses tâches. L'une des plus pertinentes dans le cas des séries temporelles est la classification des signaux de santé, et notamment de sommeil.

3.2.6 Conclusion

Nous avons présenté le contexte et avons formulé le problème d'approximation de DTW. Nous avons discuté des principales approximations de DTW et de leurs limites. ucrDTW et FastDTW permettent de réduire la complexité de l'algorithme, mais ne sont pas différentiables. À l'inverse, softDTW et LDPS permettent de différentier DTW mais conservent la complexité temporelle de l'algorithme DTW (légèrement réduite par LDPS). Ainsi, à notre connaissance, aucune approche ne satisfait les deux critères conjointement, pour produire une approximation de DTW de faible complexité et différentiable.

Nous souhaitons donc obtenir une approximation de DTW alliant faible complexité et différentiabilité, en restant fiable. Les réseaux de neurones étant une solution possible pour obtenir une telle approximation, nous avons présenté des méthodes de la littérature pour approximer des fonctions avec des réseaux pour des équations différentielles. Des approches similaires existent pour approximer les distances, en exploitant toute l'information des signaux à comparer en même temps, à l'image de l'approximation de distance de Wasserstein réalisée via un autoencodeur siamois, dont nous nous inspirons pour approximer DTW.

Dans les différents exemples précédents ([17], [21], [40]), la fidélité de l'approximation est premièrement mesurée par MSE. Cependant, notre objectif principal ne réside pas dans la valeur de sortie de DTW en elle-même. L'essentiel est notre capacité à évaluer et à hiérarchiser les similarités entre différentes séries temporelles. Donc, plutôt que de comparer les valeurs brutes de notre approximation et celles de la DTW, nous orientons notre attention sur les performances de notre approximation sur des tâches nécessitant de comparer des séries temporelles.

Nous souhaitons utiliser notre approximation de DTW pour des applications réelles, notamment la compression et la classification de signaux de santé. De fait, nous validons et comparons notre méthode via ses performances sur des tâches de comparaisons de signaux de sommeil.

3.3 Notre méthode pour approximer DTW

Dans cette section, nous décrivons les architectures que nous utilisons pour approximer l'algorithme DTW. Nous présentons une architecture siamoise, inspirée par les travaux de [40] et [100], et une directe, prédisant directement la valeur de DTW.

3.3.1 Architecture siamoise

Pour obtenir la distance entre deux séries, l'algorithme DTW recherche le chemin d'alignement entre les séries, puis calcule la somme des distances euclidiennes des points alignés. Nous souhaitons pouvoir comparer des séries temporelles grâce à notre approximation, ainsi nous n'avons pas besoin d'explicitement apprendre à déterminer le chemin d'alignement, mais simplement la valeur de distance entre les deux séries. Ainsi, nous sommes dans un cas similaire à la tâche d'approximation de la distance

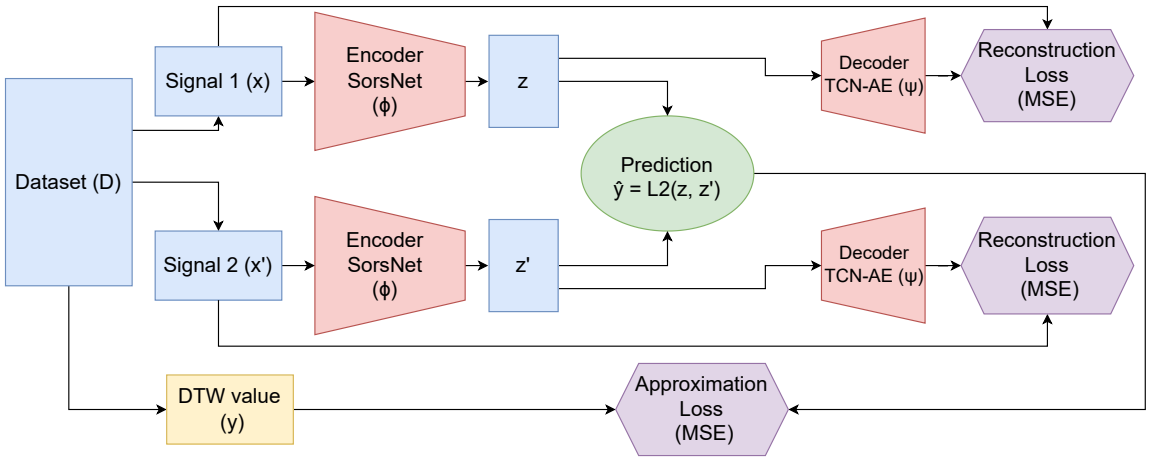


FIGURE 3.6 – Architecture globale de notre méthode. Deux signaux sont extraits du jeu de données et encodés. L'objectif est que la distance euclidienne entre les signaux encodés soit aussi proche que possible de la DTW entre les signaux originaux. Le décodeur tente ensuite de retrouver les signaux originaux à partir des représentations.

de Wasserstein, étudiée par [40] : nous souhaitons que notre modèle prédise la valeur de la distance en utilisant les deux séries temporelles en entrée. Du fait des divers avantages de l'architecture de [40] énoncés précédemment, nous choisissons d'adapter leur architecture siamoise à l'approximation de DTW.

Puisque l'objectif de [40] est d'imiter la distance de Wasserstein, les auteurs utilisent une divergence de Kullback-Leibler comme fonction de coût pour mesurer l'erreur de reconstruction commise par le décodeur. En effet, la distance de Wasserstein compare des distributions de probabilités, il est donc logique d'utiliser la KL pour comparer les éléments. À l'inverse, DTW compare directement des signaux, on utilise donc une MSE pour comparer les sorties du décodeur avec les signaux originaux. Ainsi, nous représentons dans la figure 3.6 notre approche siamoise. Nous remplaçons l'encodeur par le SorsNet [159] (que nous présenterons dans la section 3.3.3.1) et la fonction de coût de reconstruction (*Reconstruction Loss*) par la MSE. En reprenant la définition de l'encodeur ϕ et des signaux d'entrée unidimensionnels [3] $x \in \mathbb{R}^N$ et $x' \in \mathbb{R}^M$, l'encodeur projette les signaux en deux représentations $z \in \mathbb{R}^H$ and $z' \in \mathbb{R}^H$ avec H la dimension latente. Le décodeur ψ tente de reconstruire le signal d'entrée à partir de z . Nous définissons donc $\hat{x} = \psi(z)$.

En pratique, le modèle reçoit des paires de signaux $\{x^{(i)}, x'^{(j)}\}_{i,j \in 1,\dots,n}$. En nommant $y^{(i,j)}$ la valeur de DTW (l'objectif de l'approximation), on cherche à apprendre ϕ et ψ pour minimiser, avec $z^{(i)} = \phi(x^{(i)})$:

3. Nous rappelons que nous avons fixé $K = 1$. Ce choix n'impacte ici que le type d'encodeur et de décodeur choisi.

$$\min_{\phi,\psi} \underbrace{\sum_{i,j} \left| \|z^{(i)} - z'^{(j)}\|^2 - y^{(i,j)} \right|^2}_{\text{erreur d'approximation}} + \lambda \underbrace{\left(\sum_{i,j} \|\psi(z^{(i)}) - x^{(i)}\|^2 + \sum_{i,j} \|\psi(z'^{(j)}) - x'^{(j)}\|^2 \right)}_{\text{erreur de reconstruction}} \tag{3.19}$$

On souhaite ainsi apprendre les poids de l'encodeur ϕ et du décodeur ψ pour minimiser l'erreur de reconstruction (écart entre les signaux originaux et décodés) et l'erreur d'approximation (écart entre les prédictions et les valeurs réelles de DTW). λ est un hyperparamètre réel. Son objectif est d'équilibrer les fonctions de coût.

3.3.2 Procédure d'apprentissage

Nous détaillons notre procédure d'apprentissage via l'algorithme 3. L'encodeur reçoit donc des paires de signaux $\{x^{(i)}, x'^{(j)}\}_{i,j \in 1,\dots,n}$ de longueur N et M et de dimension $K = 1$. L'encodeur est utilisé deux fois : la projection d'un signal donné $x^{(i)}$ est réalisée de manière indépendante de celle de $x'^{(j)}$. Ainsi, la valeur cible est :

$$y^{(i,j)} = DTW(x^{(i)}, x'^{(j)})_{i,j \in 1,\dots,n} \tag{3.20}$$

En pratique, elle est calculée en amont de l'apprentissage pour toutes les paires de signaux considérées pour gagner du temps. Le décodeur reçoit ensuite les signaux projetés z, et tente de retrouver les signaux originaux x. De nouveau, nous utilisons simplement le décodeur deux fois. Nous obtenons l'erreur de reconstruction en comparant les signaux reconstruits aux signaux originaux selon la MSE. Nous additionnons ensuite les fonctions d'erreur pondérées par λ et nous rétro-propageons le gradient pour mettre à jour les poids de l'encodeur et du décodeur.

Algorithm 3 Boucle d'apprentissage pour chaque mini batch pour l'approximation de DTW

Entrée Mini-batch $\bar{D}$, encodeur ϕ, décodeur ψ
Sortie Mise à jour de ϕ et ψ

$Loss \leftarrow 0$
for $\forall\,(x,\, x')$ in $\bar{D}$ **do** :
  $y \leftarrow DTW(x, x')$ ▷ Pré-calculée
  $z,\, z' \leftarrow \phi(x,\, x')$ ▷ $z \in \mathbb{R}^H$
  $y_{pred} \leftarrow \|z - z'\|_2$
  $L_{encoder} \leftarrow MSE(y_{pred}, y)$
  $\hat{x},\, \hat{x}' \leftarrow \psi(z,\, z')$
  $L_{decoder} \leftarrow MSE(\hat{x}, x) + MSE(\hat{x}', x')$
  $Loss \leftarrow Loss + (L_{encoder} + \lambda * L_{decoder})$
end for
Mise à jour de ϕ, ψ selon $Loss$

Nous avons présenté le paradigme global d'approximation de DTW. Nous devons maintenant définir l'architecture de l'encodeur chargé de projeter les signaux.

3.3.3 Choix de l'encodeur

Le rôle de l'encodeur dans l'architecture est de projeter les signaux de telle sorte que leur distance euclidienne dans l'espace latent soit aussi proche que possible de leur DTW dans l'espace original. Ainsi, l'encodeur doit être capable de bien représenter les données temporelles, mais nous restons assez libres sur le choix du type de couches du réseau. Néanmoins, nous souhaitons que notre approximation puisse comparer des séries temporelles sans *padding* (ajouter des zéros au début ou à la fin du signal pour atteindre une taille donnée). En effet, DTW n'ignore pas les valeurs nulles dans son calcul. Par exemple, soit $x^{(1)} \in \mathbb{R}^N$, $x^{(2)} \in \mathbb{R}^M$ avec $M < N$. Si l'on crée $x^{(2')} \in \mathbb{R}^N$ en *paddant* $x^{(2)}$ jusqu'à N alors $DTW(x^{(1)}, x^{(2)}) \neq DTW(x^{(1)}, x^{(2')})$. Nous souhaitons donc que notre approximation imite cette propriété de DTW, et considère les zéros comme des valeurs à part entière.

Notre objectif est de plus d'utiliser l'approximation de DTW sur des données de sommeil, il est donc assez logique d'utiliser un modèle de classification de sommeil, qu'on sait capable de bien représenter les EEGs pour cette tâche. Nous souhaitons aussi une approximation rapide, avec une faible complexité liée à la longueur des séries. Ainsi, nous souhaitons éviter les approches autoregressives (qui génèrent les points des séries l'un après l'autre, en utilisant les points générés pour les suivants), qui ralentissent nécessairement l'apprentissage et l'inférence quand les séries deviennent longues.

Nous utilisons ainsi le SorsNet [159] comme encodeur, et nous le présentons dans la suite. Ce réseau satisfait tous les critères précédents, mais d'autres choix étaient envisageables, comme le TCN-AE [170]. Cependant, on obtenait des performances similaires au prix d'un temps d'inférence plus long dans nos expériences.

3.3.3.1 SorsNet

L'architecture du réseau est présentée figure 3.7. Le SorsNet est un réseau composé d'une suite de blocs, comprenant couche de convolution, *batch normalization*, ReLU comme fonction d'activation, et *dropout*. À la fin du réseau, une couche dense rassemble l'information, puis une autre réalise la classification du sommeil. Nous supprimons la dernière couche pour utiliser le SorsNet comme encodeur. Le choix des paramètres de taille de noyaux, de pas de convolution et de *stride* permettent que les sorties du réseau aient la même taille indépendamment de celle d'entrée, tant qu'elle est inférieure à 3000, qui est la taille standard d'une fenêtre de sommeil échantillonnée à 100 Hz. L'architecture entièrement convolutionnelle avant la dernière couche permet de plus d'être efficient en temps.

Le SorsNet permet donc d'obtenir un encodeur capable de bien représenter les données de sommeil, de s'adapter à diverses tailles de signaux sans devoir réapprendre certaines couches du modèle et d'être efficient en termes de temps de calcul. Après avoir présenté l'encodeur, nous présentons l'implémentation choisie pour le décodeur.

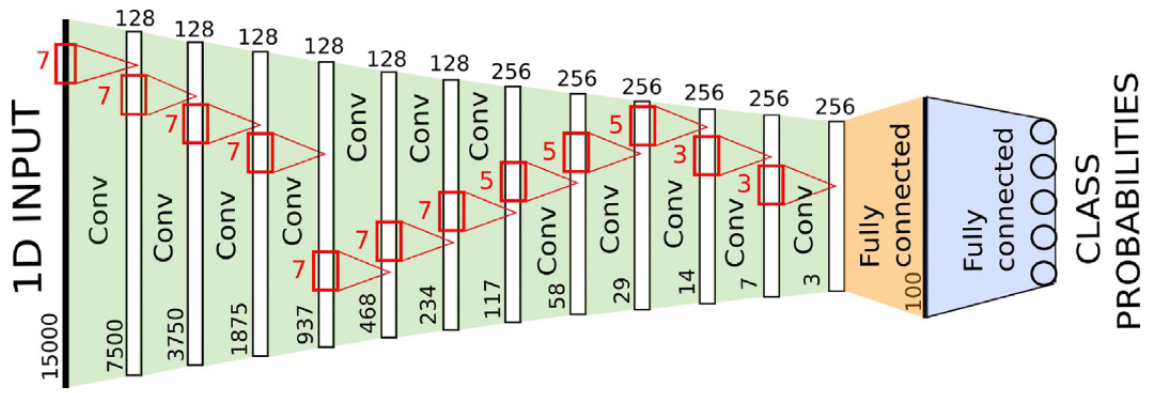


FIGURE 3.7 – Architecture du SorsNet [159].

3.3.4 Choix du décodeur

Le décodeur n'est pas obligatoire dans notre architecture, il est d'ailleurs supprimé en inférence. Néanmoins, il est utile pour régulariser l'apprentissage. En effet, son but est de reconstruire le signal, ce qui force l'encodeur à garder le maximum d'informations dans les signaux encodés. Cela aide à éviter que l'apprentissage ne s'effondre, c'est-à-dire que l'encodeur projette tous les signaux vers la même représentation et reste bloqué dans un minimum local.

L'architecture du décodeur est indépendante de celle de l'encodeur, tant qu'elle est compatible avec les dimensions des signaux projetés z. Nous choisissons d'imiter le décodeur du TCN-AE, proposé par [170]. Nous rappelons que le décodeur du TCN-AE est formé de blocs de convolutions dilatées (TCN). Ce choix est motivé par le fait que l'architecture fonctionne avec différentes tailles de signaux. En effet, le signal encodé z est d'abord ré-échantillonné pour atteindre sa taille d'origine. Cette opération n'étant pas apprise, elle fonctionne peu importe la taille des signaux. L'enchaînement de couches de convolution dilatée des blocs TCN permet de retrouver le signal original. D'autres choix de décodeurs étaient possibles : une fois les signaux encodés ré-échantillonnés, n'importe quelle architecture peut fonctionner.

3.3.5 Architecture directe

Bien que l'architecture de notre autoencodeur permette de comparer des signaux de différentes longueurs, une architecture directe pourrait mieux fonctionner sur des signaux de longueurs fixes. Pour étudier cette idée, nous introduisons également une architecture avec laquelle les deux signaux sont encodés conjointement, que nous appelons "régression directe". Nous présentons l'architecture dans la figure 3.8.

Elle prend en entrée des paires de signaux $x^{(i)}, x'^{(j)} \in \mathbb{R}^N$, les concatène pour obtenir un tenseur $X_c \in \mathbb{R}^{N\times 2}$, puis envoie X_c en entrée de l'encodeur SorsNet. Celui-ci projette X_c pour obtenir $z \in \mathbb{R}^H$. Ensuite, z est modifié par une couche dense avec *batchnorm* et activation ReLU, puis une autre couche dense réalise la prédiction de valeur de DTW. Dans ce cas, nous n'avons pas besoin d'un décodeur puisque nous obtenons directement la valeur à prédire. Tout le reste est maintenu identique à l'ar-

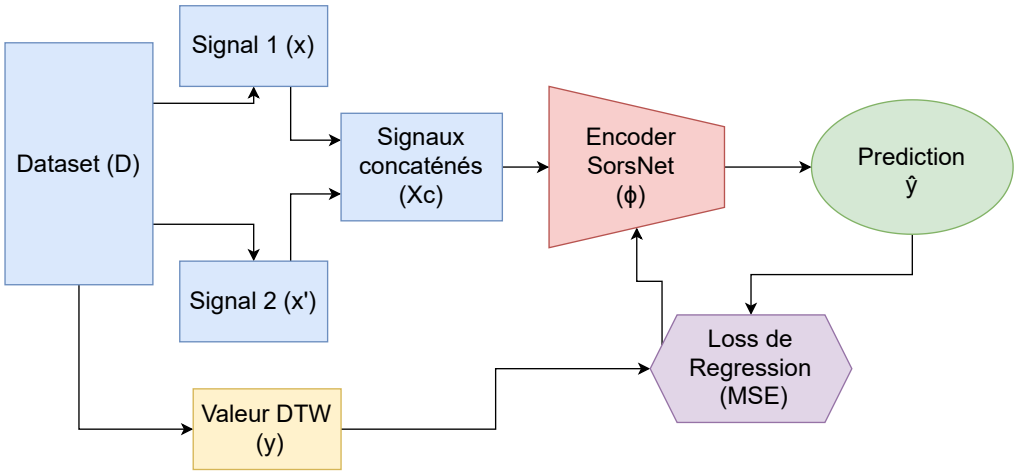


FIGURE 3.8 – Architecture globale de notre approximation directe. Deux signaux sont extraits du jeu de données puis concaténés pour obtenir l'entrée $X_c \in \mathbb{R}^{N,2}$. L'encodeur prédit ensuite directement la valeur de DTW, qui est comparée à la réalité par MSE.

chitecture siamoise. Cette architecture est plus rapide, puisque concaténer les signaux évite de les encoder deux fois comme dans l'architecture siamoise. Elle permet cependant moins d'interprétabilité des résultats, notamment de l'espace de projection des signaux, comme mentionné par [40].

3.3.6 Conclusion

Nous avons présenté deux architectures pour approximer DTW. La première utilise un autoencodeur siamois pour projeter les signaux de telle sorte que la distance euclidienne dans l'espace projeté soit proche de la DTW dans l'espace initial. Couplé à l'architecture convolutionnelle de l'encodeur, cela permet de comparer des signaux de différentes longueurs comme le fait DTW, sans *padding*. Nous avons aussi présenté une architecture qui concatène puis essaie de directement prédire la valeur de DTW. Cette architecture est plus rapide et plus simple à apprendre car elle ne nécessite pas de décodeur. Elle perd cependant en interprétabilité, et nécessite d'appliquer du *padding* si les séries sont de tailles différentes. Nos deux architectures permettent d'obtenir une approximation de DTW différentiable et dont la complexité est peu impactée par la longueur des séries temporelles. Dans la suite, nous présentons les jeux de données sur lesquels nous apprenons nos méthodes, ainsi que les expériences que nous faisons pour évaluer leurs performances.

3.4 Expérimentations

Dans cette section, nous apprenons et évaluons nos méthodes d'approximation de DTW sur des jeux de données de sommeil, et en commentons les résultats.

3.4.1 Jeu de données

Dans un premier temps, nous apprenons et évaluons notre méthode sur le jeu de données SleepEDF-78. Puisque nous apprenons uniquement à approximer la DTW, nous sommes libres d'utiliser tous les canaux des signaux de polysomnographie. Néanmoins, nous restreignons notre approximation à des signaux 1D pour faciliter la capacité de nos modèles à généraliser à d'autres jeux de données, comme introduit dans la section 3.1.2. Ainsi, pour chaque enregistrement d'une nuit d'un patient, nous séparons le signal en coupes de taille L, et choisissons aléatoirement un des canaux. Cela nous permet d'apprendre à approximer la DTW de signaux plus variés qu'en se restreignant à un type de canal, et donc d'améliorer les capacités de généralisation de notre approche. On obtient donc un jeu de données de dimension $\mathrm{Ns} \times L \times 1$, avec Ns le nombre de signaux. Le choix de restriction en 1D impacte peu l'architecture de nos méthodes. En effet, il suffit d'adapter l'encodeur de sorte qu'il prenne en entrée des signaux de la dimension souhaitée, et le décodeur de sorte qu'il reconstruise des signaux de mêmes dimensions qu'en entrée de l'encodeur. Nous séparons aléatoirement 8 patients pour former le jeu de test et 8 autres patients pour former le jeu de validation. Les 62 autres patients sont intégrés au jeu d'entraînement.

3.4.1.1 Prétraitements

Si utiliser tous les canaux permet d'améliorer la robustesse de notre modèle, cela pose un problème d'échelle : les canaux provenant de divers capteurs, ils ont des amplitudes de valeurs différentes. Cela engendre une grande amplitude des valeurs cibles de DTW et donc de la fonction de coût, rendant l'apprentissage instable. Ainsi, nous bornons par *clipping* les valeurs du jeu de données D, c'est-à-dire que nous calculons p_1 et p_{99}, respectivement les premiers et 99ème centiles, puis pour chaque valeur x_k de chaque signal $x^{(i)}$ dans D, nous appliquons :

$$x_k = \begin{cases} p_1 & \text{si } x_k < p_1 \\ p_{99} & \text{si } x_k > p_{99} \\ x_k & \text{sinon} \end{cases} \tag{3.21}$$

Ensuite, nous utilisons la normalisation min-max pour projeter les valeurs dans $[0, 1]$. Le *clipping* permet d'éviter que presque toutes les valeurs ne soient projetées proche de 0.5 à cause de valeurs aberrantes. Cette opération permet que $DTW(x, x') \in [0, L + M]\ \forall x, x' \in \mathbb{R}^{L\times 1}, \mathbb{R}^{M\times 1}$, puisque DTW revient à une somme de distances euclidiennes suivant le chemin d'alignement une fois celui-ci déterminé, et que chaque distance euclidienne est dans $[0, 1]$ du fait de la normalisation. Ainsi, nous pouvons aussi normaliser les valeurs de DTW en divisant simplement par $L + M$, et obtenir une valeur cible dans $[0, 1]$, ce qui facilite l'apprentissage du modèle.

3.4.1.2 Matrice de vérité terrain

On remplit ensuite la matrice de DTW de vérité terrain, dont chaque coefficient représente la DTW référence entre deux signaux. Pour éviter que notre modèle surapprenne les signaux d'entraînement, nous n'utilisons pas toutes les paires de signaux pour l'apprentissage. Ainsi, nous tirons au hasard Ns_{pairs} de signaux $(x^{(i)}, x'^{(j)})$ parmi

les $\frac{\mathrm{Ns}*(\mathrm{Ns}-1)}{2}$ possibles (car $DTW(x, x') = DTW(x', x)$). Nous créons donc la matrice de vérité terrain $Y_{DTW} \in \mathbb{R}^{\mathrm{Ns}\times\mathrm{Ns}}$ telle que $Y_{DTW}(i, j) = DTW(x^{(i)}, x'^{(j)})$.

Nous avons présenté les traitements nécessaires à l'apprentissage sur des données de sommeil. Dans la suite, nous donnons les détails de l'apprentissage, et notamment les hyperparamètres de nos méthodes.

3.4.2 Implémentations et apprentissage

Afin d'accélérer l'entraînement, nous limitons les signaux à une longueur $L = 1000$ en découpant aléatoirement chaque signal de longueur originale 3000. Nous sélectionnons aléatoirement $\mathrm{Ns}_{train} = 10000$ signaux pour le jeu d'entrainement et calculons la DTW de $\mathrm{Ns}_{pairs} = 10^6$ paires tirées aléatoirement parmi les $5 * 10^7$ paires possibles afin d'entraîner le modèle sur un nombre suffisant de signaux sans provoquer de sur-apprentissage. Cela permet également de limiter la durée de l'entraînement. Nous procédons de la même manière pour le jeu de validation et le jeu de test, en restreignant les valeurs pour accélérer les calculs, avec $\mathrm{Ns}_{val} = \mathrm{Ns}_{test} = 1000$ signaux et 10^5 paires tirées aléatoirement.

Nous fixons le *dropout* du SorsNet, l'encodeur de notre méthode, à 0. En effet, notre modèle sur-apprend peu et nous pouvons toujours ajouter plus de signaux si nécessaire, nous n'avons donc pas besoin de *dropout*. Nous remplaçons la couche de classification du SorsNet par une autre couche dense. Nous expérimentons avec des tailles d'espace latent $H \in [100, 250, 500, 750, 1000]$ et choisissons $H = 500$ car cela minimise l'erreur d'approximation sur le jeu de validation.

Nous utilisons le TCN-AE comme décodeur comme mentionné précédemment. Pour les blocs de convolutions dilatées, nous choisissons les ratios de dilatation (*dilation rates*) $q = [32, 16, 8, 4, 2, 1]$ et la taille du noyau $k = 20$, suivant la contribution originale [170].

Nous utilisons l'optimiseur Adam [88] avec un taux d'apprentissage de 10^{-5} pour optimiser simultanément les paramètres de l'encodeur et du décodeur. Ce choix de taux d'apprentissage relativement bas est motivé par le fait que nous cherchons à prédire une valeur exacte, et donc que des variations des poids du modèle trop importantes risquent de créer de grosses erreurs de prédiction. Nous validons expérimentalement cette décision. Nous fixons la taille de *batch* à 128 et λ à 1 : cette valeur donnait les meilleurs résultats sur le jeu de validation parmi celles testées ($[0, 0.01, 0.1, 1, 10]$). Nous entraînons le modèle pendant 50 epochs avec un *early stopping* si l'erreur de validation ne diminue pas pendant 8 epochs consécutives. Toute notre implémentation est réalisée via PyTorch [4].

Une fois nos méthodes apprises, nous les utilisons sur des tâches annexes (ou *downstream*) pour évaluer leurs performances. Nous comparons nos méthodes : la siamoise est nommée *DeepDTW Siamese*, (ou DeepDTW-S), la régression directe est nommée *DeepDTW Direct* (ou DeepDTW-D). Nous ajoutons aux comparaisons la DTW dite

4. https://pytorch.org/

"classique" (nommée simplement DTW), en utilisant l'implémentation de *tslearn*[5], ucrDTW[6] avec $r = 5\%$, FastDTW[7] avec $r = 1$ (dans cette implémentation, r n'est pas exprimé en pourcentage de la taille de la série), softDTW[8] avec $\gamma = 0.1$ et LDPS[9] en utilisant les paramètres recommandés dans la contribution originale [100] (longueurs des *shapelets* L_s respectivement égales à 15%, 30% et 45% de la longueur N des séries x, $N_{sp} = 10 * \log(N - L_s)$ éléments par sous-groupe). Les paramètres sont choisis pour obtenir un bon compromis précision / vitesse.

3.4.3 Vitesse

Dans cette partie, nous étudions la vitesse d'exécution de nos méthodes, critère important d'une bonne approximation de DTW.

Pour chaque métrique, nous tirons aléatoirement $\mathrm{Ns} = 100$ paires de signaux $(x^{(1..\mathrm{Ns})}, x'^{(1..\mathrm{Ns})})$ dans le jeu de données SleepEDF-78. Nous mesurons le temps moyen nécessaire à calculer la similarité entre $x^{(i)}$ et $x'^{(j)}, \forall i, j \in [1, \mathrm{Ns}]$. Nous faisons varier la taille des signaux (en prenant les valeurs suivantes : $[100, 500, 1000, 2000, 3000]$) pour étudier l'impact qu'elle a sur le temps de calcul. Nous présentons les résultats figure 3.9.

Avec des signaux relativement courts (moins de 1000 échantillons), nos approximations sont plus lentes que l'algorithme standard de DTW. C'est assez attendu, nos approximations étant basées sur un encodeur fait pour des signaux de 3000 points. Ainsi, peu importe la taille des signaux, toutes les couches du réseau doivent être utilisées. Puisque nous comparons les éléments 1 à 1, nous n'utilisons pas l'avantage du calcul par lot des réseaux de neurones. Notre approche directe est donc environ 2 fois plus rapide que la siamoise, car la siamoise implique d'utiliser 2 fois l'encodeur. Nous remarquons que le temps nécessaire à nos méthodes croît beaucoup plus lentement avec la taille des signaux que les autres approximations de DTW, et avec une taille de 3000, seule ucrDTW est plus rapide. En particulier à cette taille, nos méthodes sont environ 10 fois plus rapides que LDPS et 100 fois plus rapides que softDTW, qui sont les autres approximations différentiables de DTW. Nos résultats semblent de plus confirmer ceux exposés dans [181] : FastDTW est plus lent que l'algorithme original avec nos conditions d'expérimentations.

Nous montrons ainsi que même sans accélération via GPU et calcul par lots, nos méthodes sont au moins proches, voire meilleures que la majorité des approximations usuelles de DTW en termes de temps de calcul. Dans la suite, nous étudions la fidélité de nos méthodes à DTW, c'est-à-dire leur capacité à comparer les séries comme DTW.

5. https://tslearn.readthedocs.io/en/stable/user_guide/dtw.html
6. https://pypi.org/project/ucrdtw/
7. https://pypi.org/project/fastdtw/
8. https://github.com/Maghoumi/pytorch-softdtw-cuda
9. https://rtavenar.github.io/hdr/parts/02/shapelets_cnn.html

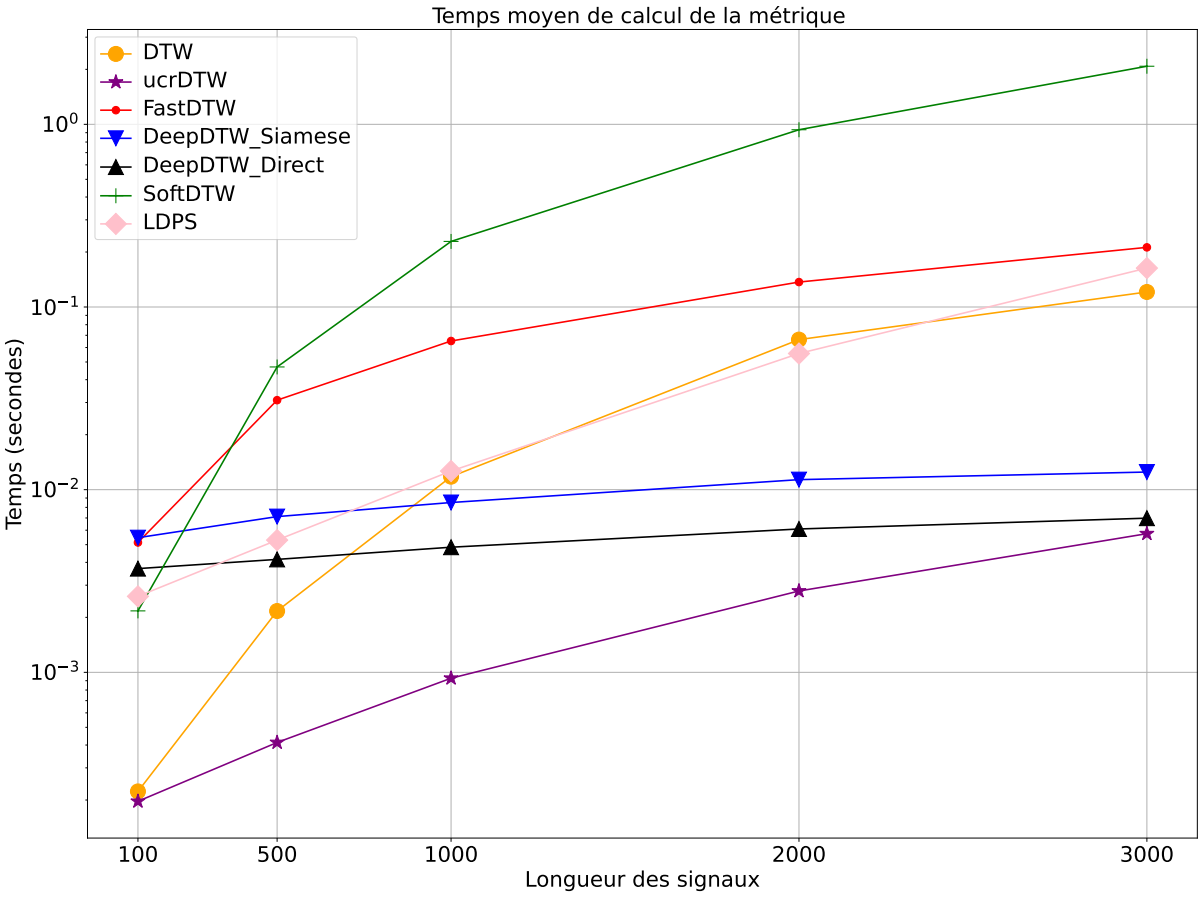


FIGURE 3.9 – Temps moyen nécessaire à calculer la métrique entre deux signaux de sommeil, en fonction de la longueur des signaux. Tous les calculs sont exécutés en Python sur CPU (Intel Xeon Silver 4112 CPU @ 2.60GHz).

3.4.4 Fidélité

Le critère le plus important d'une approximation est sa capacité à correctement imiter les résultats de l'algorithme original. Ainsi, nous évaluons notre approximation grâce à diverses tâches de comparaison et de classification de signaux. Nous comparons les résultats de notre méthode avec ceux des approximations usuelles de DTW présentées dans la section 3.2 : FastDTW, ucrDTW, softDTW et LDPS.

3.4.4.1 Plus proche voisin

DTW est couramment utilisée pour chercher des séries temporelles similaires, à l'image de [172] qui utilisent DTW pour rechercher des patients similaires selon diverses séries temporelles de capteurs médicaux.

Ainsi, nous voulons mesurer la capacité de notre approximation à trouver la série la plus similaire à une série *query* q. Nous apprenons deux instances de plus proche voisin [10], une se basant sur DTW et une sur l'approximation de DTW à évaluer, comme métrique de comparaison des éléments. Ensuite, pour chaque série *query* $q \in N_t$, nous obtenons $\hat{x}_{top1}$ le signal le plus proche de q selon notre approximation, en s'assurant que $\hat{x}_{top1} \neq q$. Pour chaque $\hat{x}_{top1}$ ainsi obtenu, nous regardons s'il fait partie des 5 plus proches voisins $[x_{top1}, .., x_{top_5}]$ de q selon l'instance basée sur DTW. Cette métrique, nommée top-5 *accuracy*, est notamment couramment utilisée sur le jeu de données ImageNet [142]. La tâche est de plus en plus dure à mesure que N_t augmente, mais aussi que les signaux se ressemblent. Pour limiter la variance de l'expérimentation, nous calculons ce score 8 fois en tirant les N_t signaux aléatoirement. Nous choisissons de plus différents N_t pour évaluer la capacité de la métrique sur différentes difficultés. Nous appliquons la standardisation (décrite dans l'équation 2.28) aux signaux quand nous utilisons ucrDTW, suivant les recommandations de la publication originale [134].

Dans un premier temps, nous utilisons le jeu SleepEDF-78 pour nos expériences. Nous utilisons la même séparation en jeu d'entraînement, jeu de validation et jeu de test que pour l'apprentissage de nos méthodes d'approximation de DTW, pour nous assurer qu'il n'y ait pas de fuite de données. Nous affichons le résultat dans le tableau 3.1.

N_t	DeepDTW-D	DeepDTW-S	ucrDTW	FastDTW	softDTW	LDPS
50	**87.3 ±4.6**	64.3 ± 10.1	39.0 ± 3.8	72.0 ± 7.7	37.3 ± 6.8	59.6 ± 4.6
100	**71.8 ±5.9**	46.2 ± 8.1	35.8 ± 3.2	66.8 ± 1.3	29.0 ± 3.9	48.8 ± 3.2
200	**64.8 ±2.8**	31.1 ± 1.9	27.0 ± 5.5	55.1 ± 3.7	21.0 ± 2.7	35.1 ± 3.6
400	**56.0 ±2.6**	23.4 ± 2.4	23.5 ± 2.7	50.3 ± 2.4	14.9 ± 1.0	23.4 ± 2.3

TABLE 3.1 – Pourcentage de fois que le signal le plus proche d'un signal donné q selon une métrique est dans le top 5 des signaux les plus proches de q selon DTW. N_t est le nombre de signaux dans le jeu de test.

Nous remarquons que nos approximations sont compétitives, si ce n'est meilleures, que les autres approximations de DTW. L'architecture siamoise est largement supérieure

10. https://scikit-learn.org/stable/modules/neighbors.html

à ucrDTW et softDTW avec un faible nombre de signaux, de 25 points de pourcentage (pp) avec 50 signaux dans le jeu, et de 11 pp avec 100 signaux par rapport à ucrDTW. L'écart se réduit en augmentant la difficulté de la tâche (9 pp d'écart avec softDTW avec 400 signaux, et similaire à ucrDTW). L'architecture siamoise obtient des résultats similaires à LDPS pour toutes les difficultés. L'architecture directe est encore meilleure : c'est la meilleure approximation, surpassant même FastDTW (de 9.7 pp avec 200 signaux). De nouveau cependant, les performances de l'approche directe se dégradent un peu plus vite à mesure que la tâche devient difficile, mais restent meilleures de 5.7 pp que FastDTW avec 400 signaux. Nous montrons ainsi que nos approximations sont capables de comparer les séries de manière similaire à DTW.

Cependant, un des principaux avantages de DTW par rapport à la distance euclidienne est sa capacité à comparer des séries temporelles de longueurs différentes. Nos approximations sont capables de le faire du fait de la construction de leurs réseaux de neurones. Nous évaluons cette propriété dans la suite.

Ainsi, nous reprenons l'expérience précédente mais modifions notre jeu de données. Nous appliquons l'*uniform sampling* en suivant [167] pour ré-échantillonner nos données de sorte qu'elles soient de taille variable $L_{us} \in [500; 1000]$. De fait, nous étudions aussi la capacité de nos approximations à imiter l'invariance au décalage temporel de DTW. Nous n'avons pas besoin d'adapter l'architecture siamoise : du fait de la définition des couches du réseau de neurones du SorsNet, tant que la série x est de longueur $L \in [500, 3000]$, elle sera projetée en z de taille $H = 500$. Ainsi, nous pouvons directement comparer deux séries de longueurs différentes. L'architecture directe nécessite d'appliquer le *padding* pour faire en sorte que la série la plus courte soit de même longueur que la plus longue avant la comparaison. Le reste des paramètres est inchangé et nous présentons les résultats dans le tableau 3.2.

N_t	DeepDTW-D	DeepDTW-S	ucrDTW	FastDTW	softDTW	LDPS
50	76.0 ± 5.4	58.7 ± 5.8	39.7 ± 2.1	**76.7 ±7.5**	34.0 ± 3.7	51.2 ± 4.5
100	53.7 ± 6.5	44.2 ± 4.1	34.2 ± 2.1	**68.7 ±5.6**	21.2 ± 9.4	33.5 ± 2.8
200	40.5 ± 2.2	32.2 ± 2.3	31.4 ± 2.6	**59.8 ±4.0**	15.0 ± 2.2	28.8 ± 4.0
400	31.8 ± 3.0	22.1 ± 1.5	26.4 ± 2.1	**55.5 ±1.4**	11.5 ± 1.4	20.9 ± 1.6

TABLE 3.2 – Pourcentage de fois que le signal le plus proche d'un signal donné q selon une métrique est dans le top 5 des signaux les plus proches de q selon DTW. N_t est le nombre de signaux dans le jeu de test. Les signaux sont ré-échantillonnés pour être de taille $L \in [500, 1000]$.

Les résultats de toutes les méthodes sauf FastDTW et ucrDTW sont inférieurs à ceux obtenus avec des signaux de longueur fixe, ce qui est attendu, la tâche étant un peu plus difficile. En particulier, notre architecture directe perd en moyenne 17 pp par rapport à la version avec des signaux de longueur fixe. Ce résultat est logique : la méthode n'est pas apprise sur des signaux de longueurs différentes, et souffre du *padding*. Notre approche siamoise est moins impactée (-2pp), montrant l'avantage de cette architecture quand les longueurs de séries varient. Nos approximations restent cependant proches de FastDTW, et largement meilleures que softDTW et ucrDTW.

N_s	DTW	DeepDTW-D	DeepDTW-S
1000	**0.445 ± 0.03 ; 0.553 ± 0.05**	0.372 ± 0.03 ; 0.511 ± 0.03	0.353 ± 0.03 ; 0.461 ± 0.02
2000	**0.456 ± 0.01 ; 0.569 ± 0.01**	0.387 ± 0.01 ; 0.503 ± 0.01	0.354 ± 0.01 ; 0.427 ± 0.01
4000	**0.495 ± 0.01 ; 0.594 ± 0.01**	0.439 ± 0.01 ; 0.569 ± 0.01	0.413 ± 0.01 ; 0.484 ± 0.01

TABLE 3.3 – Score Macro F1 et *accuracy* de la classification d'état du sommeil sur SleepEDF-78 par 1-PPV en utilisant différentes métriques. On moyenne le résultat de 3 itérations. Pour chaque itération donnée, toutes les méthodes utilisent les mêmes sous-parties du jeu de données.

Du fait des mauvais résultats d'ucrDTW et de la lenteur de softDTW nous ne les utiliserons plus pour la suite des expérimentations.

3.4.4.2 Classification via k plus proche voisin (KPPV)

Nous évaluons aussi la capacité de notre approximation à comparer les signaux dans le contexte de classification d'état du sommeil. Nous choisissons Ns signaux dans notre jeu de test de SleepEDF-78 et séparons ce jeu en un sous-jeu de support et de requête [11] en proportion 75/25. Du fait des bonnes performances du 1-PPV sur diverses tâches de classification de séries temporelles (développé dans la section 2.1.2), nous réalisons la classification via un 1-PPV, basée sur DTW, et comparons les résultats avec une classification 1-PPV basée sur nos méthodes : DeepDTW-D et DeepDTW-S. Nous obtenons un score MF1 et un score *accuracy*. En raison de la lenteur de l'expérimentation (l'évaluation de 1-PPV nécessite de nombreux calculs de DTW), nous nous limitons à des nombres de signaux Ns relativement bas. Pour limiter la variance induite par ce faible nombre de signaux, nous répétons l'expérience 3 fois, en tirant aléatoirement les Ns signaux à chaque fois.

Nous présentons les résultats dans le tableau 3.3. Sans surprise, le 1-PPV basé sur DTW obtient les meilleurs résultats, indépendamment du nombre de signaux dans le jeu. Avec 4000 signaux, 1-PPV$_{DTW}$ obtient un MF1 de 0.495, largement au-dessus de l'aléatoire, mais bien en-dessous des performances à l'état de l'art (Sleepy-Co [93] obtient un score MF1 de 0.83). C'est en partie dû au fait que la lenteur des calculs de DTW empêche d'utiliser des grands jeux de données, mais aussi que la classification de sommeil est une tâche demandant un modèle puissant et le 1-PPV semble ne pas suffire, peu importe la métrique utilisée. Néanmoins, nos approximations de DTW performent convenablement. À 4000 signaux, notre approche directe est notamment à environ 6 pp de DTW. L'approche siamoise obtient un score un peu moindre, environ 2 pp en dessous de l'approche directe, illustrant la supériorité de l'approche directe pour la classification 1-PPV quand les signaux sont de même longueur.

Ainsi, pour un problème de recherche de séries similaires à une série *query*, nos approximations se comportent de façon similaire à la DTW référence. En particulier, DeepDTW-D surpasse FastDTW sur des séries de taille fixe, mais souffre du *padding* sur des données de taille variable. La classification 1-PPV illustre la difficulté de discriminer les signaux de sommeil sans création de caractéristiques et avec des jeux de

11. L'algorithme KPPV n'apprenant pas, on utilise les termes de support et requête. Ils sont néanmoins analogues aux jeux d'entraînement et de validation.

données restreints.

3.4.5 Différentiabilité

Puisque notre approximation se base sur des réseaux de neurones, elle est différentiable. On peut donc l'utiliser directement comme fonction de coût pour apprendre une tâche. Dans cette section, nous utilisons notre approximation directe comme métrique pour apprendre des prototypes de signaux de sommeil et ainsi apprendre la classification d'état du sommeil.

[33] ont proposé une méthode pour apprendre des prototypes spécifiques à chaque classe afin de classer les séries temporelles. Pour un jeu de données D, les auteurs proposent d'apprendre autant de prototypes $p \in \mathbb{R}^L$ qu'il y a de classes k dans le jeu de données. Une fois les prototypes appris, la classification se fait simplement en affectant à une série x la classe du prototype p_k le plus proche.

Les prototypes sont appris en calculant la DTW entre un signal donné $x \in D$ et chaque prototype p_k correspondant à chaque classe k. On prédit alors que l'élément x est de classe k si k minimise $DTW(x, p_k)$. Une *Cross-Entropy* entre les différentes prédictions d'un *batch* et la vérité terrain permet d'évaluer les prédictions. On modifie ainsi la valeur des prototypes de telle sorte que p_k soit proche de x si $y = k$, et loin sinon. Pour que les prototypes représentent bien leur classe, on souhaite aussi minimiser $d_k = DTW(x_{y=k}, p_k)$ avec $x_{y=k}$ indiquant que la série x est de classe k. La fonction de coût globale est alors :

$$\mathcal{L} = CE(\hat{y}, y) + \lambda * d_k \tag{3.22}$$

λ est de nouveau un réel servant à équilibrer les valeurs des erreurs. On le détermine en fonction du score sur le jeu de validation. Comme l'objectif est d'apprendre les prototypes de bout en bout, pour contourner la non-différentiabilité de la DTW les auteurs de [33] choisissent, à l'image de DTWNet, de différencier la DTW en utilisant la forme déterminée le long du chemin d'alignement, c'est-à-dire la somme des distances euclidiennes des points appariés. Nous nommons cette méthode « méthode du chemin »pour simplifier.

Puisque le reste du fonctionnement de la classification par prototypes ne nécessite pas d'obtenir le chemin d'alignement, mais simplement d'obtenir la distance entre les signaux, nous pouvons remplacer la méthode du chemin de [29] par notre approximation et évaluer les performances. Nous appliquons donc la méthode de classification par prototypes à la classification de sommeil sur le jeu de données SleepEDF-78. Pour accélérer les expériences, nous restreignons la taille des jeux à 20000 signaux en apprentissage et 5000 en validation. Nous présentons les résultats figure 3.10, en choisissant l'*accuracy* comme métrique pour suivre la publication originale [33]. L'*accuracy* sur le jeu de validation de la méthode se basant sur notre approximation de DTW pour comparer les signaux et les prototypes est significativement meilleure que celle se basant sur la méthode du chemin utilisant DTW et FastDTW pour obtenir le chemin d'alignement. Ainsi, les prototypes appris sont plus discriminants et représentent mieux leur classe. De nouveau, les résultats de notre méthode et de LDPS sont proches, bien

que notre méthode soit légèrement meilleure. Cela illustre qu'utiliser des approximations différentiables de DTW permet de meilleures performances que la méthode du chemin.

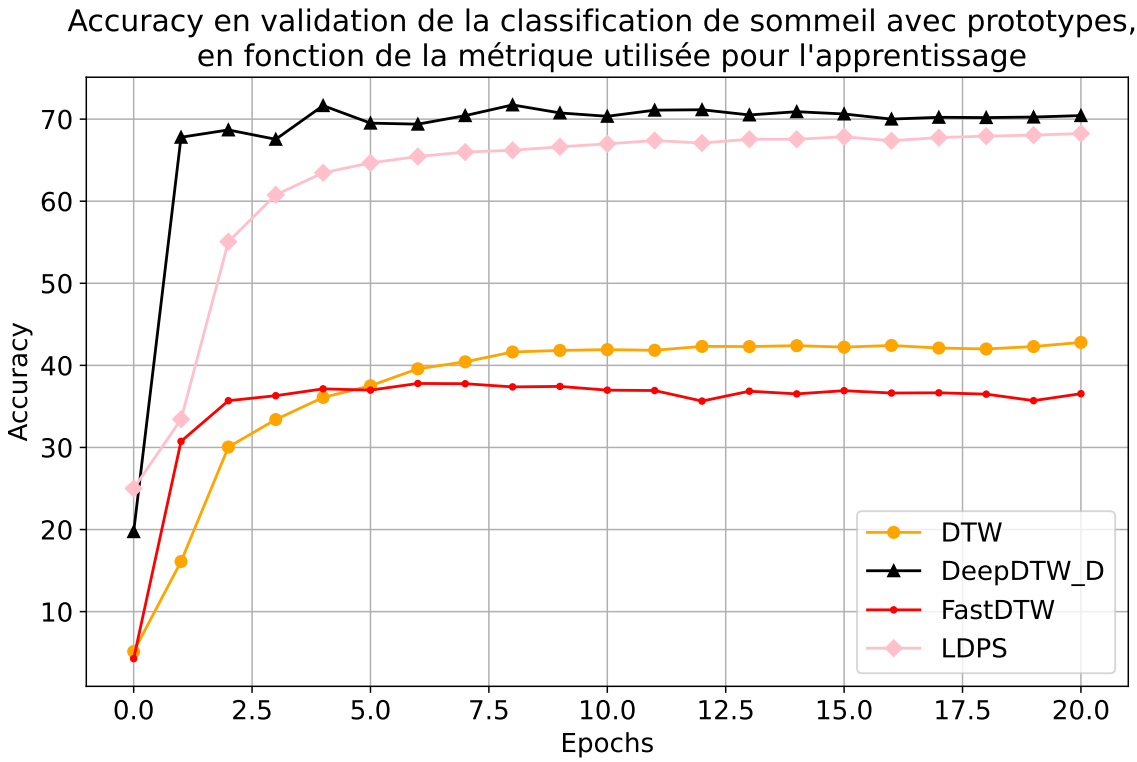


FIGURE 3.10 – *Accuracy* de la classification par prototype sur le jeu de validation pendant l'apprentissage des prototypes, en fonction de la métrique utilisée pour comparer les signaux et les prototypes. DTW et FastDTW sont utilisés pour obtenir le chemin nécessaire à la méthode de Cai et al [29].

3.4.6 Conclusion

Nous avons montré que nos méthodes d'approximation sont capables de bien comparer les séries temporelles, et peuvent donc être utilisées comme métrique pour des algorithmes de classification. Nos méthodes sont de plus rapides, avec un temps d'exécution évoluant peu selon la taille des signaux, et sont utilisables en tant que fonction de coût pour un apprentissage bout-en-bout. Ces résultats sont néanmoins obtenus sur un seul jeu de données. Dans la suite, nous évaluons nos approximations sur un autre jeu de données de sommeil.

3.5 Application à d'autres jeux de données

Nous avons démontré que notre approximation est rapide, utilisable comme fonction de coût et fidèle à l'algorithme DTW original via des tâches de classification et de requêtage sur le jeu de données SleepEDF-78. La validation des deux premiers critères est indépendante du jeu de données, tant que l'on reste sur des données de sommeil

N signaux	Appris sur SHHS	Appris sur SleepEDF-78
50	84.3 ± 5.0	84.7 ± 5.1
100	75.3 ± 2.9	74.0 ± 3.1
200	61.2 ± 6.2	60.2 ± 5.4
400	48.8 ± 4.1	48.4 ± 2.0

TABLE 3.4 – Score de recherche de plus proche voisin sur SHHS, comparant notre architecture directe apprise sur SHHS avec celle apprise sur SleepEDF.

de taille similaire à SleepEDF-78. Néanmoins, il est important d'évaluer la capacité de notre approximation à obtenir de bonnes performances sur des jeux de données similaires pour augmenter la facilité d'adoption des modèles par les experts en médecine du sommeil. En effet, les données réelles pourraient être obtenues via des capteurs légèrement différents par exemple. Nous évaluons donc la fidélité de notre approximation en reproduisant les expérimentations précédentes sur le jeu de données SHHS, qui est un jeu de données massif regroupant diverses sources d'enregistrement du sommeil. Il est décrit en détail section 2.2.1.1 et est notamment intéressant du fait du nombre significatif de patients dans le jeu (5791).

3.5.1 Recherche de séries similaires

Nous reproduisons l'expérience de recherche de séries similaires décrite section 3.4.4.1 : l'objectif est, pour une série *query*, de trouver la série la plus proche selon DTW. Nous apprenons notre modèle d'approximation sur les données SHHS comme expliqué dans la section 3.4.2 : nous créons la matrice de référence en calculant la DTW référence de morceaux de signaux de canaux choisis aléatoirement, puis apprenons l'approche siamoise et l'approche directe à prédire cette valeur référence. Nous comparons cette fois les performances sur SHHS de notre approximation avec architecture directe apprise sur le jeu de données SHHS avec celle apprise sur le jeu de données SleepEDF-78, sans aucun *finetuning* sur SHHS. L'objectif de cette expérience est d'identifier la capacité de notre approximation à bien comparer des signaux de sommeil d'un autre jeu de données. Nous utilisons le même prétraitement qu'auparavant, pour que nos approximations comparent des séries prenant des plages de valeurs identiques (entre 0 et 1).

Nous présentons les résultats dans le tableau 3.4. Les scores obtenus sur SHHS par la méthode apprise sur SHHS et celle apprise sur SleepEDF-78 sont quasiment identiques pour toutes les difficultés, ce qui illustre la capacité de notre approximation à bien comparer des séries temporelles, même si elles sont issues d'un jeu de données différent (en conservant néanmoins le même prétraitement). Cela nous permet d'envisager utiliser nos approximations pour des tâches de compression ou de classification de signaux de sommeil sur divers jeux de données.

N_s	DTW	DeepDTW-S	DeepDTW-D
1000	**0.4 ± 0.01 ; 0.484 ± 0.01**	0.346 ± 0.01 ; 0.449 ± 0.02	0.334 ± 0.02 ; 0.427 ± 0.02
2000	**0.428 ± 0.01 ; 0.51 ± 0.02**	0.366 ± 0.02 ; 0.473 ± 0.01	0.367 ± 0.02 ; 0.455 ± 0.01
4000	**0.541 ± 0.01 ; 0.587 ± 0.01**	0.487 ± 0.01 ; 0.552 ± 0.01	0.486 ± 0.01 ; 0.529 ± 0.01

TABLE 3.5 – Macro F1 et *accuracy* de classification de sommeil sur SHHS en ayant appris sur SleepEDF-78. Les classifieurs sont des 1-PPV basés sur les différentes métriques. Chaque résultat affiché est la moyenne de 3 itérations, avec l'écart type associé.

3.5.2 Classification 1-PPV

De la même façon, nous reproduisons l'expérience décrite dans la section 3.4.4.2 sur le jeu de données SHHS : nous utilisons donc diverses métriques comme base pour réaliser la classification via un 1-PPV. Nous utilisons nos approximations apprises sur le jeu de données SleepEDF-78, de nouveau sans aucun *finetuning*. Nous présentons les résultats dans le tableau 3.5.

Les résultats sont globalement similaires à ceux obtenus sur le jeu de données SleepEDF-78 [12] : DTW est le meilleur classifieur pour tous les nombres de signaux considérés selon le MF1 et l'*accuracy*. Les métriques que nous proposons ont un score MF1 d'environ 6 pp inférieur à DTW. Ce résultat est d'autant plus intéressant que l'apprentissage de DeepDTW a été conduit sur un autre jeu de données. Ceci illustre la généricité de notre approche et la capacité de nos métriques à bien comparer des séries de sommeil provenant de différents jeux de données.

3.5.3 Classification de sommeil via prototypes

Enfin, nous reproduisons l'expérience de classification de sommeil via apprentissage de prototypes, cette fois sur le jeu de données SHHS. Nous gardons tous les paramètres de l'expérience sur SleepEDF-78 : les méthodes DeepDTW et LDPS ne sont pas *finetunées* sur le jeu SHHS. Nous présentons les résultats figure 3.11. De nouveau, notre approximation permet d'apprendre des prototypes discriminant mieux les classes que les méthodes du chemin basées sur FastDTW et DTW. L'écart entre notre méthode et LDPS est plus grand que celui obtenu durant la même expérience sur le jeu SleepEDF-78, ce qui illustre que notre méthode généralise mieux à de nouvelles données.

3.6 Synthèse des résultats

Nous résumons dans cette section les comparaisons effectuées entre les différentes approximations de DTW. Nous affichons cette synthèse dans le tableau 3.6. Nous rangeons les méthodes selon leur vitesse d'exécution, leur différentiabilité, leur fidélité (en prenant en compte les résultats des expériences de plus proche voisin), leur perte de fidélité sur des signaux de longueurs différentes (que nous résumons par l'écart moyen arrondi entre les scores de PPV obtenus sur les signaux de longueurs fixes et ceux obtenus sur les signaux de longueurs variables et que nous notons PFLD) et leur capacité

12. décrits dans le tableau 3.3

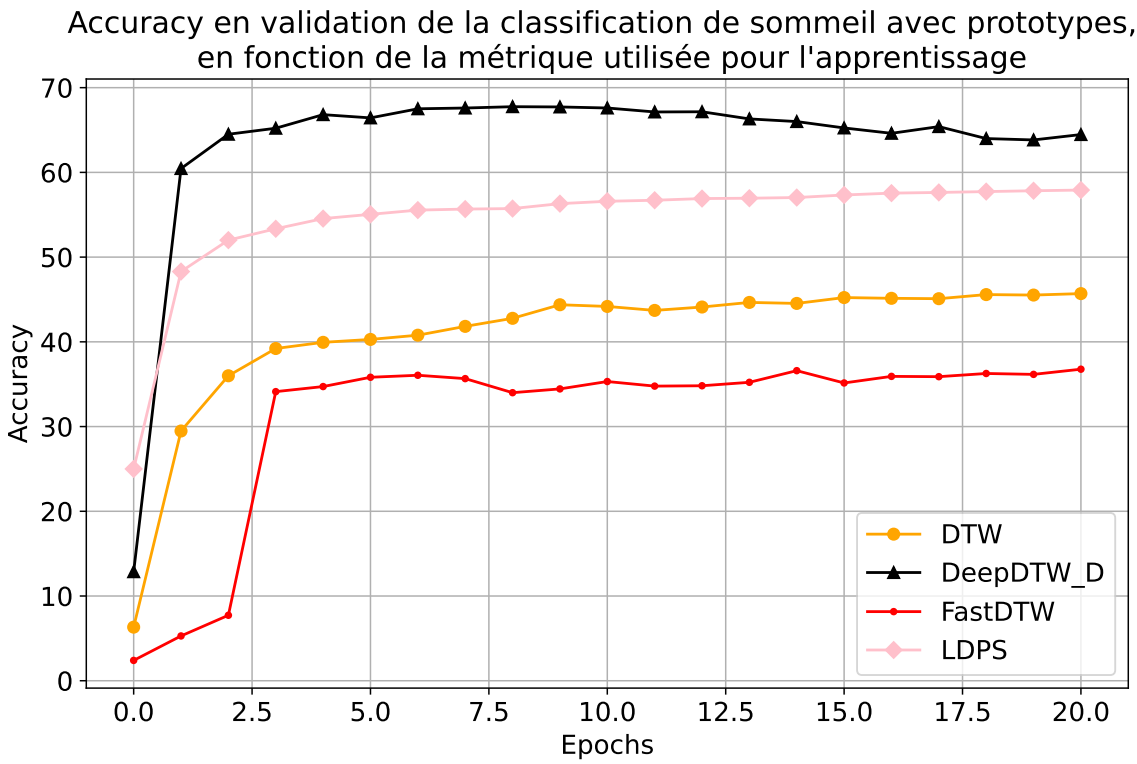


FIGURE 3.11 – *Accuracy* de la classification par prototype sur le jeu de validation de SHHS pendant l'apprentissage des prototypes, en fonction de la métrique utilisée pour comparer les signaux et les prototypes.

Méthode	Vitesse	Différentiable	Fidélite	PFLD	Généralisation
DTW	4	Non	Référence	Référence	Pas d'app.
ucrDTW	1	Non	5	Aucune	Pas d'app.
FastDTW	6	Non	2	Aucune	Pas d'app.
SoftDTW	7	Oui	6	-5pp	Pas d'app.
LDPS	5	Oui	3*	-7pp	Bonne
DeepDTW-S	3	Oui	3*	-2pp	Bonne
DeepDTW-D	2	Oui	1	-19pp	Bonne

TABLE 3.6 – Synthèse des résultats des comparaisons d'approximation de DTW développées dans ce chapitre. Les valeurs de vitesse et de fidélité sont les rangs des méthodes. Les résultats de fidélité de LDPS et DeepDTW Siamese étant très similaires, les méthodes sont classées troisièmes égalité (3*).

à généraliser (en prenant en compte les expériences sur le jeu SHHS sans *finetuning* ; nous notons « Pas d'app. »quand les méthodes ne requièrent pas d'apprentissage et donc généralisent parfaitement). Globalement, ces résultats illustrent que notre approche directe semble être le meilleur compromis pour approximer DTW si les signaux sont de taille fixe. L'approche siamoise est le meilleur choix sinon.

3.7 Conclusion du chapitre

De nombreuses applications nécessitent de comparer des séries temporelles. L'algorithme DTW permet de faire cette comparaison, même si les séries sont de différentes tailles ou décalées temporellement. Cependant, l'algorithme est de complexité quadratique et non différentiable. Nous avons présenté une méthode permettant d'approximer DTW via un réseau de neurones, rendant l'approximation fiable, rapide et différentiable. Nous avons présenté et comparé deux architectures permettant de réaliser cette approximation : une siamoise et une directe. L'approche siamoise consiste à créer un espace latent dans lequel la distance euclidienne imite la DTW. Un tel espace est obtenu en entraînant un autoencodeur : l'objectif de l'encodeur est de représenter les signaux dans cet espace et celui du décodeur est de reconstruire les signaux originaux à partir des signaux encodés. L'approche directe concatène simplement les signaux pour prédire directement la valeur de DTW. Nous avons montré que nos méthodes permettent de comparer les séries temporelles de manière similaire à DTW. L'approche directe est plus rapide que la siamoise et plus fidèle à DTW, au prix d'une perte d'interprétabilité de l'espace latent et d'une nécessité de *padding* si l'on compare des séries de tailles différentes.

Nous aimerions pouvoir comparer des séries temporelles multidimensionnelles. Il n'y a pas besoin de modifier l'architecture globale de notre système, mais une recherche du type d'encodeur optimal serait nécessaire, pour utiliser la possible redondance d'informations entre les différentes dimensions.

Nous avons montré que nos approximations peuvent être utilisées dans un entraînement de bout en bout, qu'elles sont plus rapides et plus fidèles à DTW que d'autres approximations usuelles et qu'elles obtiennent une bonne performance dans les tâches de classification des séries temporelles. Enfin, nous avons également démontré que les modèles appris généralisent à d'autres jeux de données. Nous présentons dans la suite de ce document l'utilisation de cette approximation pour la compression de signaux de sommeil.

Chapitre 4

Compression de signaux de sommeil via approximation de DTW

Sommaire

4.1 Introduction 98
4.1.1 Contexte et motivations 98
4.1.2 Définition du problème 99
4.1.3 Métriques 100
4.2 État de l'art 101
4.2.1 Approches basées sur des dictionnaires 102
4.2.2 Approches basées sur la transformation par fonction 104
4.2.3 Approches par autoencodeurs 106
4.2.4 Conclusion 109
4.3 Méthode proposée pour compresser les signaux 110
4.3.1 Architecture générale 110
4.3.2 Choix de la fonction de coût 112
4.3.3 Architecture de l'encodeur-décodeur 112
4.3.4 Apprentissage 113
4.4 Résultats expérimentaux 113
4.4.1 Comparaisons avec les approches non paramétriques 113
4.4.2 Classification sur les données reconstruites 114
4.5 Conclusion 119

Compression de signaux de sommeil via approximation de DTW

4.1 Introduction

4.1.1 Contexte et motivations

Nous avons présenté dans les chapitres précédents diverses tâches liées aux séries temporelles. En particulier, les enregistrements de sommeil permettent d'apprendre automatiquement la classification d'état du sommeil et aident à diagnostiquer des troubles d'insomnie. Si la classification d'état du sommeil est effectuée sur des séries de 30 secondes et peut fonctionner avec un seul canal, d'autres études peuvent nécessiter de plus longues fenêtres d'enregistrements multidimensionnels. Cela peut entraîner des difficultés de stockage, et des lenteurs, aussi bien pour le transfert de données que pour leur utilisation par des modèles d'apprentissage automatique. La compression est habituellement utilisée pour résoudre ce problème très courant dans le traitement du signal.

De nombreux algorithmes existent pour restreindre la taille des signaux, comme la compression MP3 pour l'audio, ou JPEG pour les images. Les approches de compression des séries temporelles sont usuellement séparées en 3 catégories [34] : les approches basées sur des dictionnaires comme [86], qui sélectionne des morceaux de signaux particuliers appelés atomes avant d'encoder les autres morceaux de signaux comme une combinaison linéaire des atomes similaires ; les approches qui transforment le signal dans une nouvelle représentation puis sélectionnent les composantes les plus importantes du signal projeté, comme DCT [109] ; et les approches apprenant les poids d'un autoencodeur de telle sorte que les signaux encodés soient de plus petite dimension que les signaux originaux, comme [145]. Les systèmes sont évalués selon l'erreur de reconstruction, le facteur de compression atteignable et la vitesse d'exécution de l'algorithme.

En particulier, il est courant dans la littérature de valider la performance des systèmes de compression via des métriques de reconstruction comme l'erreur quadratique moyenne (MSE), le rapport signal/bruit (SNR) et le score de qualité (QS) comme fait par [179], [42] ou [98]. Ces métriques sont cependant basées sur la distance euclidienne : or, les résultats de classification 1-PPV mentionnés dans la section 2.1.2.4, dans laquelle nous illustrions qu'un 1-PPV basé sur DTW obtient de meilleurs scores en classification qu'un 1-PPV basé sur la distance euclidienne, suggèrent que la distance euclidienne n'est pas suffisante pour discriminer les séries temporelles. Ainsi, différentes publications utilisent en complément une métrique sur une tâche de classification pour valider le système de compression (notamment [145], [51], [31]). Nous soutenons qu'une telle approche, axée sur une tâche finale comme la classification, est nécessaire. En effet, pour les applications de classification du sommeil par exemple, après avoir été compressé pour la transmission, nous voulons que le signal reconstruit conserve le maximum d'informations discriminantes permettant de bien classifier l'état du sommeil.

Nous n'utilisons pas le score de classification des stades de sommeil comme fonction de coût pour optimiser les poids du modèle de compression. Une telle approche risquerait de générer des signaux reconstruits qui, bien qu'efficaces pour la classification, seraient probablement inutilisables pour toute autre tâche, limitant ainsi la généricité du système de compression. Les signaux de sommeil reconstruits seraient de plus difficilement interprétables par un médecin, car ne "ressemblant" pas visuellement au signal original. Or, l'interprétabilité des décisions prises par le modèle est un facteur important pour l'adoption des modèles d'apprentissage automatique dans le domaine de la santé [162]. À la place, nous utilisons une approximation de DTW comme fonction de coût pour la reconstruction. Cette méthode nous permet d'apprendre les poids d'un encodeur et d'un décodeur capable de compresser et de reconstruire les signaux en prenant en compte leur information temporelle.

Dans ce chapitre, nous définissons le problème de compression sous l'angle de l'apprentissage, puis présentons les trois principaux types d'approches proposées par la communauté pour compresser les séries temporelles : par dictionnaire, par transformation par fonction et par autoencodeur. Nous discutons aussi du très important choix de la fonction de coût de reconstruction pour les approches basées sur des autoencodeurs. Nous présentons ensuite l'autoencodeur que nous utilisons pour apprendre la compression de signaux, dont l'encodeur et le décodeur sont basés sur le TCN. Nous utilisons nos approximations de DTW comme fonction de coût. Nous présentons ensuite nos résultats d'expérimentations sur des jeux de données de sommeil, et étudions l'impact de diverses fonctions de coût sur le score de classification. Nous montrons qu'utiliser notre approximation directe de DTW améliore le score de classification du sommeil sur les données compressées puis reconstruites, en particulier quand le facteur de compression augmente.

4.1.2 Définition du problème

La compression de données est définie dans [149] et [34] comme le processus de conversion d'un flux de données d'entrée (le flux source ou les données brutes originales) en un autre flux de données (la sortie, ou le flux comprimé) de taille plus petite. L'objectif de la compression est d'éliminer la redondance tout en ayant une grande puissance descriptive. Le processus de décompression, complémentaire à la compression, cherche à reconstruire le flux de données original à partir de sa représentation compressée.

Les algorithmes de compression sont usuellement catégorisés selon les critères suivants [34] :

- Avec ou sans apprentissage : les algorithmes avec apprentissage apprennent leurs paramètres sur un jeu de données. Ils sont donc dépendants du jeu en question, et peuvent ne pas généraliser à d'autres types de données.
- Avec ou sans perte : si les signaux reconstruits après compression et décompression sont exactement identiques aux signaux originaux, la compression est dite sans perte. C'est l'objectif de tout système de compression, mais n'est atteignable que si l'entrée contient de l'information redondante.
- Symétrique ou non symétrique : un algorithme est dit symétrique si les opérations effectuées par le décodeur sont identiques (mais en ordre inverse) à celles faites

par l'encodeur.

Dans le cas particulier de la compression de séries temporelles, un algorithme de compression (encodeur, ϕ) prend en entrée une série temporelle $x \in \mathbb{R}^{N \times K}$ et retourne sa représentation compressée $z \in \mathbb{R}^{H}$, avec $H < N*K$. À partir de la représentation compressée, en utilisant un décodeur ψ, il est possible de reconstruire la série temporelle originale : $\hat{x} = \psi(z) = \psi(\phi(x))$. Si $\hat{x} = x$ alors l'algorithme est sans perte.

4.1.3 Métriques

Les performances d'un système de compression de séries temporelles sont évaluées selon trois critères : la vitesse, le facteur de compression et la précision. La vitesse d'exécution est importante dans le contexte d'applications en temps réel, comme la détection de fatigue durant la conduite [8]. Les travaux présentés dans la suite ne sont pas dans ce contexte, donc nous nous concentrerons sur le facteur de compression et la précision.

Le facteur de compression mesure le ratio de taille entre le signal original et le signal compressé. Bien qu'en pratique, la taille des données soit définie par l'espace mémoire requis, nous la définissons dans ce chapitre par le nombre d'échantillons N multiplié par le nombre de dimensions K de la série x. Cette approche simplifie la comparaison entre différentes méthodes de compression, car elle nous affranchit des variations dues à la représentation réelle des séries, qui pourraient dépendre du langage de programmation employé.

Le facteur de compression est ainsi défini par :

$$CF = \frac{N * K}{H} \tag{4.1}$$

Naturellement, plus CF est grand, plus la compression est importante. Dans la littérature [145], [31], le ratio de compression CR est parfois exprimé à la place de CF, et est défini par :

$$CR = 100(1 - \frac{H}{N * K}) \tag{4.2}$$

Pour faciliter les comparaisons entre les différentes architectures, nous utiliserons :

$$CF = \frac{1}{1 - \frac{CR}{100}} \tag{4.3}$$

Cela permet de convertir les ratios en facteur de compression si nécessaire.

La qualité de la reconstruction est quantifiée via plusieurs métriques développées dans la suite, mesurant l'écart entre le signal original x et celui reconstruit $\hat{x}$, avec $x, \hat{x} \in \mathbb{R}^{N \times K}$. Dans la suite et pour les raisons développées aux chapitres précédents (les performances d'algorithme de classification du sommeil en 1D sont généralement très proches, si ce n'est équivalentes à celles sur des signaux multidimensionnels [126]), nous choisissons $K = 1$, sans perdre de généralité dans ce chapitre.

Le rapport signal/bruit (SNR), qui est le rapport de puissance entre la force du signal et le niveau de bruit, s'exprime par :

$$SNR_{DB} = 10 * log_{10}(\frac{P_x}{P_{\hat{x}}}) \tag{4.4}$$

avec

$$P_x = \frac{\sum_{i=1}^{N} x_i^2}{N} \tag{4.5}$$

Et l'écart quadratique moyen (MSE), on le définit par :

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{4.6}$$

On peut aussi utiliser la racine de l'écart quadratique moyen (RMSE).

Cet écart peut être normalisé, on le nomme alors PRD et il est défini par :

$$PRD(x, \hat{x}) = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{\sum_{i=1}^{N} x_i^2}} \times 100 \tag{4.7}$$

Le PRD, exprimé en pourcentage, est parfois aussi nommé distorsion.

Le facteur de compression est pris en compte via le *Quality Score* :

$$QS = \frac{CF}{PRD} \tag{4.8}$$

Ainsi, les métriques CF, SNR, PRD et QS sont les plus usuellement utilisées pour comparer les performances de compression [34], [145].

En fonction des applications, certaines métriques peuvent être plus importantes que d'autres, par exemple, si les données doivent être précisément reconstruites, on accordera plus d'importance à la RMSE. Si l'on souhaite cependant comparer les performances de modèles de compression de manière plus globale, on s'intéressera plus particulièrement au *Quality Score* à l'image de [183], qui comprime des signaux ECG, ou de [42], qui applique la compression à des données ECG, EMG et EEG conjointement. Ces métriques restent cependant largement basées sur des comparaisons élément à élément, ce qui peut ne pas être suffisant pour comparer des signaux temporels. Nous utiliserons donc, à l'image de [145], [51], [31], un score de classification sur les données reconstruites en plus de ces métriques pour évaluer notre système de compression.

4.2 État de l'art

Les algorithmes de compression sont séparés en 3 catégories distinguant mieux leur fonctionnement que les critères précédents, comme indiqué par les travaux de [34] :

— Les approches basées sur des dictionnaires recherchent des morceaux répétés dans les séries puis les représentent par des clés de dictionnaire.

— Les approches par transformations transforment la série temporelle en un ensemble de coefficients via des projections adaptées.

— Les approches par autoencodeurs paramétriques apprennent les paramètres d'un encodeur et d'un décodeur pour compresser et reconstruire les séries temporelles.

Nous présentons ces différentes approches en détail dans cette section.

4.2.1 Approches basées sur des dictionnaires

La compression de séries temporelles par dictionnaire **D** consiste à représenter les séries temporelles à l'aide d'un ensemble réduit d'éléments de base appelés "atomes". Ces atomes sont des segments de séries temporelles que l'on peut combiner de différentes manières pour approximer ou reconstruire la série temporelle originale.

Pour l'illustrer de façon basique, nous pouvons imaginer une série temporelle où chaque mesure est accompagnée d'un attribut indiquant la ville dans laquelle elle a été prise [1]. Plutôt que de stocker directement le nom de chaque ville, on peut créer un dictionnaire : {0 : "New York", 1 : "San Francisco", 2 : "Los Angeles", ...}. La colonne de la ville dans l'ensemble de données n'aurait alors besoin de stocker que les indices correspondants [0, 1, 1, 2, ...].

Plus généralement, nous présentons les différentes étapes nécessaires à la compression par dictionnaire dans la suite. Ces étapes peuvent varier en fonction des particularités des méthodes proposées [86], [103], [91]. Nous prendrons en exemple les travaux de [86] au vu de leurs performances pour illustrer le principe de l'approximation par dictionnaire.

La première étape consiste à créer un dictionnaire composé d'un ensemble d'atomes, qui sont des morceaux de séries temporelles ou des fonctions de base qui peuvent être combinées pour approximer d'autres séries temporelles. Dans le cas de [86], les séries $x \in \mathbb{R}^N$ sont séparées en N_m morceaux $x^{(t)}$ dans des fenêtres de temps prédéfinies. On calcule la matrice de corrélation $M_c \in \mathbb{R}^{N_m \times N_m}$ de tous les morceaux entre eux, puis on obtient pour chaque $x^{(t)}$ la somme de ses corrélations $S_{x^{(t)}} \in \mathbb{R}^{N_m}$ avec tous les autres éléments. On choisit les N_{atm} morceaux $x^{(t)}$ les moins corrélés, c'est-à-dire avec les plus faibles $S_{x^{(t)}}$ comme atomes (ou clés du dictionnaire).

Chaque segment de la série temporelle est ensuite approximé en utilisant une combinaison linéaire d'atomes présents dans le dictionnaire. Cette combinaison est dite « sparse »(ou creuse) car elle utilise un nombre minimal d'atomes pour représenter fidèlement le segment. On appelle cette étape encodage sparse. Les atomes choisis ne sont pas nécessairement orthogonaux (l'information contenue par un atome n'est pas forcément unique) : avoir plusieurs atomes redondants peut améliorer la flexibilité de la représentation [86].

Le problème d'encodage *sparse* est formulé comme un problème d'optimisation dont le but est de minimiser l'erreur de reconstruction sous contrainte de sparsité, c'est-à-dire

1. https://www.timescale.com/blog/time-series-compression-algorithms-explained/

en utilisant un nombre minimal d'atomes :

$$\hat{\gamma} = \arg\min_{\gamma} \|x - \mathbf{D}\gamma\|_2^2 \quad \text{tel que} \quad \|\gamma\|_0 \leq N_{atm} \tag{4.9}$$

ou sous contrainte d'erreur de reconstruction :

$$\hat{\gamma} = \arg\min_{\gamma} \|\gamma\|_0 \quad \text{tel que} \quad \|x - \mathbf{D}\gamma\|_2^2 \leq \epsilon \tag{4.10}$$

avec ϵ le seuil d'erreur de reconstruction, γ l'ensemble des atomes et N_{atm} le nombre maximal d'atomes.

L'optimisation sous contrainte de sparsité vise ainsi à limiter le nombre d'atomes et donc augmenter le facteur de compression, quand celle sous contrainte d'erreur a pour principal objectif d'avoir une reconstruction fidèle. Ces deux problèmes sont NP-difficiles [46], mais peuvent être approchés, notamment par *Orthogonal Matching Pursuit* (OMP) [121], une heuristique gloutonne pour rapidement obtenir un ensemble d'atomes qui approxime les morceaux de séries, avec leur poids correspondant. On décrit son fonctionnement dans l'algorithme 4. À chaque itération, on sélectionne l'atome $\mathbf{D}_j \in \mathbb{R}^{L_{atm}}$ le plus proche (selon une métrique) de la série *query* x. Cela nécessite une métrique de comparaison de séries temporelles, qui peut être la distance euclidienne, DTW, ou d'autres, comme présentées dans la section 2.1.2.3. Dans l'algorithme OMP, on recherche l'atome ayant le plus grand produit scalaire avec le signal *query* x, alors que [86] utilise le coefficient de Pearson. Le coefficient α_j représente la contribution de l'atome $\mathbf{D}_j$ à la reconstruction de x.

Algorithm 4 Encodage du dictionnaire sparse, basé sur OMP. Extrait de [86].

Entrée Série *query* x, dictionnaire d'atomes $\mathbf{D}$, nombre d'atomes N_{atm}
Sortie Ensemble d'atomes Ω, coefficients α

$r \leftarrow x$
$\Omega \leftarrow \{\emptyset\}$
$j \leftarrow 0$
while $card(\Omega) \leq N_{atm}$ **do** : ▷ selection du prochain atome D_j
  $\mathbf{D}_j \leftarrow \arg\max_{j \notin \Omega} \frac{r^T \mathbf{D}_j}{||r|| * ||\mathbf{D}_j||}$
  $\Omega \leftarrow \Omega \cup \{j\}$
  $\alpha \leftarrow (\mathbf{D}_\Omega^T \mathbf{D}_\Omega)^{-1} (\mathbf{D}_\Omega^T q)$ ▷ Maj de Ω et de α
  $r \leftarrow x - \mathbf{D}_\Omega \alpha$
  $j \leftarrow j + 1$ ▷ Maj des erreurs r
end while

Au lieu de stocker les segments eux-mêmes, on stocke les clés des atomes qui les approximent le mieux, ainsi que les coefficients α de la combinaison linéaire. Cela permet de réduire considérablement la quantité de données à stocker.

On reconstruit la série en remplaçant les morceaux compressés par les atomes les plus proches selon la métrique. Le facteur de compression et l'erreur de reconstruction

sont dépendants du nombre d'atomes choisis : un grand nombre d'atomes permet une reconstruction fidèle, mais augmente la taille du dictionnaire et donc baisse le facteur de compression. Choisir de grands atomes permet d'en choisir moins, mais augmente l'erreur de reconstruction. Dans leur méthode CORAD, [86] fixent la taille des atomes à 20 indépendamment de la taille des séries et choisissent le nombre d'atomes pour ne pas dépasser un seuil d'erreur de reconstruction calculé par MSE (approche contrainte par erreur).

Si les séries temporelles comportent peu de redondance, ou que le volume de données ne permet pas de créer un dictionnaire suffisant pour bien représenter les différentes séries, on peut utiliser des opérateurs pour directement transformer et réduire le nombre d'échantillons des séries temporelles. On décrit ces approches dans la suite.

4.2.2 Approches basées sur la transformation par fonction

Les approches basées sur la transformation par fonction consistent à projeter la série temporelle $x = [x_1, x_2, ..., x_N] \in \mathbb{R}^N$ pour obtenir une liste de coefficients $c \in \mathbb{R}^H$ la représentant. Le facteur de compression est donc simplement défini par $CF = \frac{N}{H}$.

Différents ensembles de fonctions sont couramment utilisés pour réaliser la projection : *Discrete Cosine Transform* (DCT) [163], *Discrete Wavelet Transform* [155] et *Discrete Fourier Transform (DFT)* [38].

Nous présentons les définitions de ces méthodes dans la suite.

Discrete Cosine Transform :

Plusieurs versions de la DCT existent. La plus commune, DCT-II ou simplement DCT, transforme les valeurs x_i de la série x de la manière suivante :

$$c_k = \sum_{i=0}^{N} x_i \cdot \cos\left[\frac{\pi}{N}\left(i + \frac{1}{2}\right)k\right] \tag{4.11}$$

avec $k \in [0, N-1]$. Pour compresser, on sélectionne les $H \in [1, N-1]$ coefficients les plus importants. En pratique, on met les $N - H$ coefficients non sélectionnés à 0.

On obtient la reconstruction $\hat{x}$ en appliquant la DCT-III, qui est l'opération inverse de la DCT-II, c'est-à-dire :

$$\hat{x}_k = c_0 + 2 * \sum_{i=1}^{N-1} c_i \cos\left[\frac{\pi(2k+1)i}{2N}\right] \tag{4.12}$$

La DCT est notamment utilisée par [109] pour compresser des signaux de capteurs de station spatiale.

Discrete Fourier Transform :

De la même manière, la série temporelle peut être transformée via la transformée de Fourier. En pratique, l'algorithme *Fast Fourier Transform* [23] est utilisé, ainsi :

$$c_k = \sum_{j=1}^{N} x_j \exp(\frac{-2i\pi kj}{N}) \tag{4.13}$$

et on obtient le signal reconstruit $\hat{x}$ via la transformée inverse (iFFT) :

$$\hat{x}_k = \sum_{j=1}^{N} c_j \exp(\frac{2i\pi kj}{N}) \tag{4.14}$$

Discrete Wavelet Transform :

L'analyse de séries temporelles par *wavelets* (ou ondelettes) se base sur des fonctions qui, comme des vagues, débutent et finissent en zéro, avec des oscillations [34]. À chaque fonction ondelette est associée une fonction d'échelle. Différents types de *wavelets* existent, par exemple : Haar, Daubechies, biorthogonale, coiflets ou symlets. Les plus communes sont les *wavelets* de Daubechies [45].

Dans le cas des *wavelets* de Daubechies, les fonctions d'échelle et d'ondelettes sont représentées par les coefficients d'un filtre de convolution. Les ondelettes de Daubechies sont formées de sorte à posséder le nombre maximum de moments dissipant ρ pour un support donné. Un nombre de moments dissipant ρ indique que la fonction d'échelle est suffisante pour entièrement représenter un polynôme de degré $\rho - 1$, c'est-à-dire que les coefficients de l'ondelette sont nuls. On désigne les ondelettes de Daubechies par deux noms : soit par la taille du support N_{sup} (le nombre de coefficients de la fonction d'ondelette, égal à celui de la fonction d'échelle), on utilise alors dN_{sup} ; soit par le nombre de moments dissipant ρ, on note alors dbρ. Puisque pour les ondelettes de Daubechies, $N_{sup} = 2\rho$, les notations sont facilement interchangeables.

Pour compresser une série $x \in \mathbb{R}^N$ via des ondelettes, on applique le filtre de convolution d'échelle puis celui d'ondelette à x. Cela donne deux représentations, usuellement notées respectivement cA et cD, avec $cA, cD \in \mathbb{R}^{\frac{N}{2}}$. On obtient la reconstruction en appliquant les filtres inverses. En pratique, cA contient suffisamment d'informations pour correctement reconstituer $\hat{x}$, on abandonne donc cD, ce qui permet d'obtenir un facteur de compression de 2. Pour augmenter ce facteur, on peut appliquer le filtre d'échelle en cascade, jusqu'à obtenir $cA_n \in \mathbb{R}^{\frac{N}{2^n}}$, résultat de n applications du filtre, avec $CF = \frac{N}{2^n}$.

Le choix optimal du type d'ondelette est dépendant de la tâche et des données considérées. [53] applique la DWT puis l'algorithme de Run Length Encoding (RLE) pour compresser des séries ECG issues du jeu MIT-BIH [111]. L'ondelette choisie est la symlet7, ce qui permet d'obtenir un score $QS = 6.02$ avec un facteur de compression $CF = 9.75$.

Dans [94], différentes ondelettes sont évaluées selon leur capacité à créer des caractéristiques pour classifier des séries temporelles. L'ondelette db1, ou ondelette de Haar,

obtient d'excellentes performances malgré sa simplicité (seulement deux coefficients). Ainsi, nous l'utiliserons pour compresser via DWT comme base de comparaison.

La compression par DWT et celle par DFT sont comparées dans [80] sur des signaux jouets : la meilleure méthode dépend alors du type de signaux et du type de vague utilisée par DWT. Les compressions par DCT, DFT et DWT (en utilisant différentes ondelettes) sont comparées sur le jeu MIT-BIH par [135]. La meilleure approche est la compression par DCT (CF = 6.58, QS = 0.69), légèrement meilleure que FFT (CF = 5.31, QS = 0.55). La meilleure approche par DWT utilise l'ondelette coiflets4 (CF = 3.76, QS = 0.45). Ces résultats restent cependant assez proches et limités : le facteur de compression évalué est faible. Pour ces raisons, nous comparerons DCT, DWT et FFT dans la suite de ce chapitre.

Nous avons vu dans les chapitres précédents le principe de fonctionnement des autoencodeurs et comment ils peuvent être utilisés pour approximer des fonctions. Les méthodes par transformation par fonction peuvent être vues comme des autoencodeurs, l'encodeur étant dans l'exemple du cas de la DFT la fonction FFT, et le décodeur la fonction iFFT. Dans ce cas, l'encodeur et le décodeur sont non paramétriques. Via des réseaux de neurones, on peut apprendre les paramètres de l'encodeur et du décodeur, permettant de mieux représenter le jeu de données étudié.

4.2.3 Approches par autoencodeurs

Les autoencodeurs sont très adaptés à la compression grâce à leur architecture : avec $x \in \mathbb{R}^N$, $z = \phi(x)$, $z \in \mathbb{R}^H$ le signal projeté par l'encodeur, il suffit de choisir $H < N$ pour obtenir un système de compression. La décompression est naturellement effectuée par le décodeur pour obtenir $\hat{x} = \psi(z)$.

La performance du système de compression dépend alors du choix de quatre paramètres : l'architecture de l'encodeur ϕ, celle du décodeur ψ (souvent symétrique à celle de l'encodeur, à l'image d'UNet [139]), le facteur de compression (qui peut influer sur l'architecture de l'encodeur), et la fonction de coût de reconstruction qui compare l'entrée x au signal reconstruit $\hat{x}$ et qui met à jour les poids de ϕ et de ψ.

4.2.3.1 Architecture de l'encodeur et du décodeur

Les différents types d'architectures d'encodeurs et de décodeurs adaptés aux séries temporelles ont été présentés dans la section 2.1.3.3. Nous rappelons ainsi que les principales architectures adaptées sont basées sur des réseaux récurrents, des couches de convolution ou des modules d'attention. Dans cette section, nous présentons diverses architectures utilisées pour la compression. Le choix de l'architecture étant dépendant du type de série temporelle étudiée, nous restreignons notre étude aux signaux de santé.

[145] utilise un autoencodeur basé sur des couches denses (développées dans la section 2.1.3.2) pour compresser des signaux EEG et EMG issus du jeu de données DEAP [89]. Le facteur maximal de compression obtenu est de 10, ce qui est assez limité, et la reconstruction est imparfaite (PRD = 12, QS = 0.83). Cette performance mitigée est

en partie attribuable aux limitations des architectures basées sur des couches denses lorsqu'il s'agit de traiter des séries temporelles. En effet, ces architectures ne prennent pas explicitement en compte les relations temporelles entre les valeurs successives de la série, ce qui est un inconvénient majeur pour une reconstruction précise.

Ainsi, [183] propose de remplacer les couches denses par des couches de convolution 1D pour compresser des signaux ECG. La compression est effectuée par une suite de blocs composés d'une couche de convolution, de *BatchNorm* et de *MaxPooling*, avec une ReLU comme fonction d'activation. La décompression est effectuée en alternant des couches de convolution 1D et des couches de sur-échantillonnage pour progressivement reconstruire la série compressée. Cela permet d'augmenter le facteur de compression à 32.25 avec une reconstruction fidèle (PRD = 2.73, QS = 11.85). On nomme cette approche CAE. [4] utilisent une architecture similaire pour compresser des signaux EEG, atteignant un facteur de compression de 62.5 et une faible erreur de reconstruction (PRD = 1.3, QS = 48.1) sur le jeu de données BCI-IV-2a [27].

En utilisant une seule couche de *BatchNorm* et de *MaxPooling* pour la compression, ainsi que des couches de convolution transposées (ou déconvolution) plutôt que la combinaison convolution et sur-échantillonnage pour la décompression, et la fonction *Tanh* comme activation, la méthode proposée par [179] permet d'obtenir un facteur de compression de 300. Ainsi, les signaux sont représentés par $z \in \mathbb{R}^1$. Cette forte amélioration du facteur de compression est cependant au prix d'une reconstruction moins précise (PRD = 8, QS = 37.5). On nomme cette approche SCAE. En se basant sur une architecture similaire et en utilisant l'information de 8 canaux conjointement, [51] atteignent un ratio de compression de 1600 en gardant une bonne reconstruction (PRD = 31.4, QS = 50.9) sur le jeu de données EMG-8 [5]. Néanmoins, sur des signaux unidimensionnels, le facteur de compression maximal atteint est de 256, avec une erreur de reconstruction similaire (PRD = 29.8), résultant en un QS moindre (QS = 8.6).

Comme présenté dans la section 2.1.3.3, les réseaux récurrents permettent de bien représenter la temporalité dans les séries. Ainsi, [43] ajoutent une couche contenant 8 cellules LSTM à la fin d'un encodeur similaire à celui proposé par [179]. Ils utilisent cette architecture pour compresser des signaux ECG (jeu de données MIT-BIH [111]). L'utilisation de LSTM limite le facteur de compression maximal atteignable (CF = 64) mais permet d'obtenir une reconstruction plus précise (PRD = 4.1, QS = 15.61). L'intérêt des réseaux récurrents reste limité dans la compression : en enlevant la couche LSTM, les auteurs obtiennent des performances très similaires (PRD = 4.16, QS = 15.38), démontrant que l'utilisation des couches récurrentes n'est pas toujours pertinente pour la compression de signaux de santé. On nomme cette approche CDAE-LSTM.

En se basant sur les travaux de [51], [42] utilisent un encodeur basé sur une suite de convolutions 1D avec différentes tailles de noyau pour compresser conjointement des signaux EEG, EMG et ECG (les trois signaux sont compressés par trois branches indépendantes du réseau). Les représentations sont concaténées pour obtenir le signal compressé z. z est ensuite séparé en 3, puis chaque partie est décompressée par le

décodeur, basé sur une suite de convolutions transposées qui reprend les tailles de noyau de l'encodeur, de manière symétrique. On obtient alors les trois signaux reconstitués. Même sur des signaux unidimensionnels, cette architecture atteint un facteur de compression très haut (500), tout en maintenant d'excellentes performances en termes d'erreur de reconstruction (PRD = 3.62, QS = 138.1). On nomme cette approche M-DDCAE.

Les méthodes de [135], [53], [183], [179], [43] et [42] sont apprises et évaluées sur le jeu de données MIT-BIH [111] et sont donc directement comparables. On présente les résultats de cette comparaison dans le tableau 4.1. Les méthodes utilisant des autoencodeurs permettent d'atteindre des facteurs de compression significativement supérieurs à ceux obtenus par approximation de fonction (500 contre 9.75), tout en gardant une reconstruction plus proche (selon la métrique PRD), résultant en des scores de qualité QS bien meilleurs (138.1 pour M-DDCAE contre 6.02 pour DWT + RLE). Ils nécessitent cependant un apprentissage, et ne sont donc a priori fonctionnels que sur le jeu de données sur lequel ils ont appris. En particulier pour les autoencodeurs, l'utilisation de convolutions 1D en compression, et de convolutions transposées en décompression permet d'obtenir le meilleur compromis entre facteur de compression atteignable et qualité de la reconstruction. Le facteur de compression est simplement limité par la taille des signaux : les représentations compressées ne comportent qu'un ou deux coefficients.

Méthode	PRD	CF	QS
DCT [135]	9.53	6.58	0.69
DWT (coif4) [135]	8.28	3.76	0.45
FFT [135]	9.34	5.31	0.69
DWT + RLE [53]	1.62	9.75	6.02
CAE [183]	2.73	32.35	11.85
SCAE [179]	8	300	37.5
CDAE-LSTM [43]	4.1	64	15.61
M-DDCAE [42]	3.62	500	138.1

TABLE 4.1 – Performances de différentes architectures de compression sur le jeu de données MIT-BIT. [42] utilisent des fenêtres plus longues ($N = 1000$).

Toutes les méthodes basées sur des réseaux de neurones utilisent cependant la MSE comme fonction de coût, et évaluent leurs résultats via des métriques basées sur cette mesure d'erreur (PRD, RMSE, QS). Cela ne permet pas d'évaluer la capacité des systèmes de compression à garder les informations permettant d'identifier les classes des séries temporelles. Cependant, minimiser la MSE ne garantit pas nécessairement que les informations essentielles pour la classification des séries temporelles sont préservées. En d'autres termes, une faible MSE peut être obtenue sans nécessairement conserver les caractéristiques du signal qui sont cruciales pour des tâches spécifiques, comme la classification, comme nous le montrerons au travers de nos expérimentations dans la suite de ce chapitre.

4.2.3.2 Le choix de la fonction de coût de reconstruction

Pour obtenir une reconstruction prenant en compte les caractéristiques discriminantes des signaux, il faut choisir une fonction de coût de reconstruction adaptée. Dans ce but, [143] ajoute deux termes à la fonction de coût : le premier basé sur le coefficient de Pearson (*pcc*), décrit dans l'équation 2.10, et le second sur la différence entre l'autocorrélation du signal original et du signal reconstruit. Puisque $pcc(x, x) = 1$, on crée la fonction de coût basée sur ce coefficient par :

$$\mathcal{L}_{pcc}(x, \hat{x}) = MSE(1, pcc(x, \hat{x})) \tag{4.15}$$

L'objectif est de forcer le réseau à reconstruire des signaux suivant les formes et la structure du signal original.

L'autocorrélation est la convolution d'un signal en utilisant ce même signal comme noyau. Elle est définie par :

$$r_{xx}(l) = \sum_{k=l}^{N} x_k x_{k-l} \tag{4.16}$$

En pratique, cela revient à rechercher des motifs de longueur l répétés dans un signal x. Ainsi, on peut évaluer la capacité du système à reproduire les motifs en comparant l'autocorrélation du signal reconstruit avec celle du signal original. On définit donc, en fixant préalablement l :

$$\mathcal{L}_{ac}(x, \hat{x}) = MSE(r_{xx}, r_{\hat{x}\hat{x}}) \tag{4.17}$$

Et on obtient la fonction de coût finale :

$$\mathcal{L}(x, \hat{x}) = MSE(x, \hat{x}) + \mathcal{L}_{pcc}(x, \hat{x}) + \mathcal{L}_{ac}(x, \hat{x}) \tag{4.18}$$

On nomme cette méthode *Physics Informed* suivant la contribution originale [143]. Cette modification permet par exemple une augmentation de 78% du SNR en apprenant un autoencodeur convolutionnel sur un jeu de données de défaut de roulement [2]. L'amélioration de l'*accuracy* sur les signaux reconstruits est cependant plus mesurée et difficile à évaluer car peu commentée par les auteurs : la métrique principale utilisée pour comparer les approches étant le SNR. Cette approche est à notre connaissance la seule adaptation de la fonction de coût de reconstruction à des séries temporelles.

4.2.4 Conclusion

Alors que les méthodes par dictionnaire demandent un temps de préparation significatif pour le calcul des proximités entre les fragments du signal, elles offrent l'avantage d'une grande efficacité en phase de décompression. Cependant, le choix de la taille et du nombre d'atomes représente un compromis entre le facteur de compression et l'erreur de reconstruction.

Ainsi, on peut plutôt choisir de projeter les séries temporelles via des opérateurs comme DCT, DWT ou FFT pour représenter la série en coefficients et la compresser avant

2. https://engineering.case.edu/bearingdatacenter/download-data-file

de reconstruire la série via l'opération inverse. Cette compression crée peu d'erreur avec des facteurs de compression faibles (inférieurs à 10) et ne requiert pas d'apprentissage, mais obtient de moins bonnes performances qu'en apprenant la fonction de compression et de décompression via un autoencodeur basé sur des réseaux de neurones. Ceux-ci permettent d'obtenir une compression adaptée au jeu de données, avec de hauts facteurs de compression, en particulier en utilisant des architectures convolutionnelles.

Si la fonction de coût choisie est généralement la MSE, l'intérêt de l'adapter au type de données est illustré par [143], qui ajoute un terme de corrélation et de similarité d'autocorrélation pour mieux reproduire les motifs caractéristiques des séries temporelles. Suivant ce constat, nous proposons d'utiliser DTW comme fonction de coût de reconstruction. Nous forçons ainsi l'autoencodeur à reconstruire des signaux en prenant en compte la temporalité de ceux-ci, mais ne le pénalisons pas si les signaux originaux et reconstruits présentent des motifs légèrement décalés. Puisque DTW n'est pas différentiable, nous utilisons nos approximations présentées au chapitre précédent.

Enfin, nous soutenons que les performances dans une tâche *downstream* sur les données reconstruites devrait être la métrique prépondérante. En effet, dans tous les systèmes de compression, le but est de faciliter et d'accélérer le transfert et le stockage de données. Néanmoins, la tâche initiale pour laquelle les données ont été obtenues reste l'objectif, et la validité de la compression devrait être évaluée selon cet objectif, ce que nous ferons dans la suite de ce chapitre, en choisissant la classification de sommeil comme objectif.

4.3 Méthode proposée pour compresser les signaux

4.3.1 Architecture générale

Algorithm 5 Boucle d'apprentissage du système de compression à chaque mini-batch pour une métrique donnée.

Entrée Mini-batch $\bar{D}$, encodeur ϕ, décodeur ψ, fonction de coût $\mathcal{L}$
Sortie Mise à jour de ϕ, ψ

for $\forall\,(x)$ in $\bar{D}$ **do** :
  $z \leftarrow \phi(x)$ ▷ $z \in \mathbb{R}^H$
  $z_u \leftarrow upsampling(z)$ ▷ $z_u \in \mathbb{R}^N$
  $\hat{x} \leftarrow \psi(z_u)$
  $Loss \leftarrow \mathcal{L}(\hat{x}, x)$
end for
Mise à jour de ϕ, ψ selon $Loss$

Dans cette section, nous expliquons notre méthode pour apprendre la compression de signaux. L'architecture globale de notre *pipeline* de classification avec compression est présentée dans la figure 4.1. Chaque signal $x \in \mathbb{R}^N$ du jeu de données est d'abord encodé par l'encodeur ϕ pour obtenir la représentation compressée (ou code) $z \in \mathbb{R}^H$. On simule ensuite un transfert vers la base de données stockant les codes. Pour obtenir le signal reconstruit, on sur-échantillonne le code pour obtenir à nouveau la taille

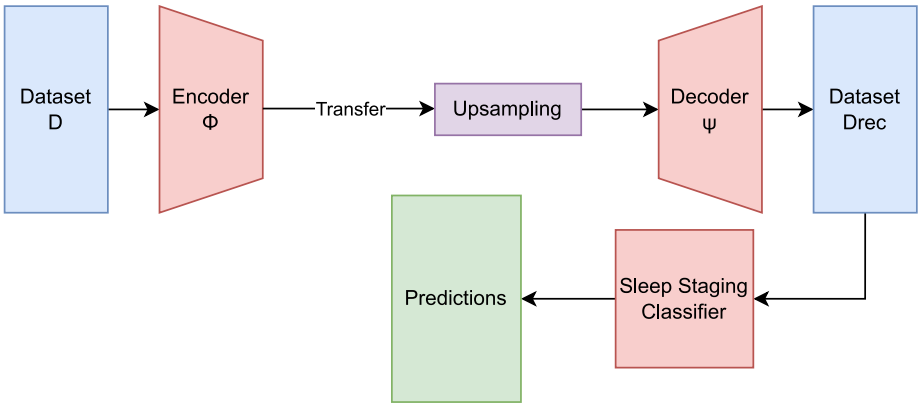


FIGURE 4.1 – Schéma du *pipeline* de classification de données compressées puis reconstruites.

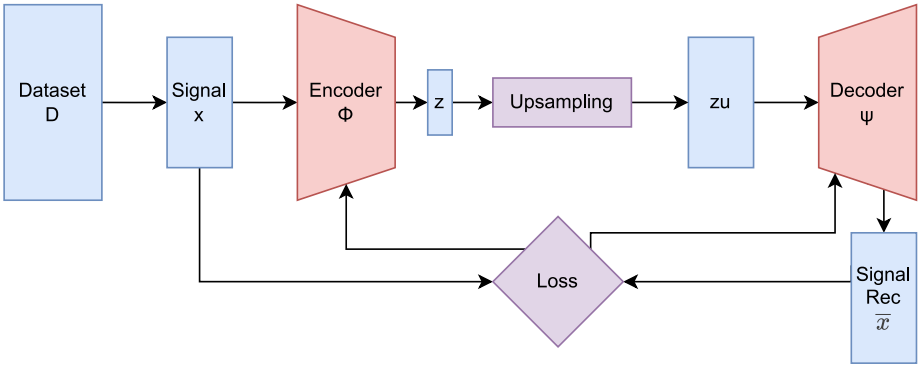


FIGURE 4.2 – Zoom sur l'architecture du système de compression.

initiale de $x \in \mathbb{R}^N$ puis le décodeur ψ tente de reconstruire le signal original. On obtient alors le jeu reconstitué D_{rec}. Ensuite, chaque signal dans D_{rec} est envoyé à un classifieur pour effectuer la tâche finale voulue : ici la classification d'état du sommeil.

Le système de compression est détaillé dans la figure 4.2. Un signal $x \in \mathbb{R}^N$ est extrait du jeu de données pour être compressé par l'encodeur ϕ. Pour rétablir le signal original, z est sur-échantillonné jusqu'à sa taille d'origine, puis envoyé au décodeur ψ . Une fonction de coût compare le signal décodé $\hat{x} \in \mathbb{R}^N$ au signal original pour calculer l'erreur de reconstruction. Via une descente de gradient, on met alors à jour les poids de l'encodeur et du décodeur.

Les différentes étapes de l'apprentissage de la compression et de la reconstruction des signaux sont détaillées dans l'algorithme 5. Un signal $x \in \mathbb{R}^N$ est extrait du jeu de données et encodé en $z \in \mathbb{R}^H$ par l'encodeur ϕ. Il est ensuite sur-échantillonné en $z_u \in \mathbb{R}^L$ puis décodé par ψ pour obtenir $\hat{x}$. On obtient l'erreur de reconstruction via la fonction de coût de reconstruction $\mathcal{L}$ entre x et $\hat{x}$ pour mettre à jour les poids de ϕ et ψ.

4.3.2 Choix de la fonction de coût

Nous avons présenté précédemment l'importance de la fonction de coût de reconstruction dans l'apprentissage d'un système de compression. Puisque nous voulons compresser et décompresser des signaux de santé, nous utilisons DTW pour comparer les signaux originaux et reconstruits. Cependant, nous ne pouvons pas utiliser DTW comme tel car l'algorithme n'est pas différentiable. À la place, nous avons recours à des approximations de DTW que nous avons présentées dans le chapitre précédent, à savoir l'architecture siamoise *DeepDTW Siamese* et directe *DeepDTW Direct*. Pour ne pas perturber l'apprentissage du système de compression et éviter des fuites de données, nous figeons les poids de ces approximations de DTW. Autrement dit, lors de l'apprentissage, ces poids demeurent constants, de sorte que les mises à jour via descente de gradient soient effectuées uniquement sur les poids de l'encodeur ϕ et du décodeur ψ du système de compression. L'objectif de l'apprentissage est alors d'ajuster les poids de ψ et ϕ suivant :

$$\min_{\phi,\psi} \quad DeepDTW(x, \psi(\phi(x)) \tag{4.19}$$

avec *DeepDTW* nos approximations de DTW présentées au chapitre précédent.

Dans le chapitre précédent, nous avons montré que l'architecture directe (*DeepDTW Direct*) obtenait de meilleurs résultats, notamment sur la tâche de recherche de séries similaires (les résultats sont présentés dans la section 3.4.4.1). Cependant, nous comparons quand même les deux architectures. En effet, nous n'utilisons plus DTW pour comparer et ranger des séries, mais pour quantifier la qualité de la reconstruction d'un signal. Ainsi, l'architecture siamoise, qui applique la même projection au signal original x qu'au signal reconstruit $\hat{x}$, permet de bien comparer l'information présente dans les deux signaux.

4.3.3 Architecture de l'encodeur-décodeur

De nouveau, le choix de l'encodeur est important pour obtenir une bonne compression. Nous avons ici besoin d'un modèle capable de bien modéliser les séries temporelles, pour en garder l'information importante lors de la compression, mais aussi d'être capable de retrouver le signal initial. Nous souhaitons aussi que le taux de compression soit facilement modifiable pour simplifier les applications concrètes du système. Enfin, nous envisageons d'utiliser le compresseur pour de très longues séries temporelles : nous souhaitons éviter les architectures auto-régressives, dont le temps d'exécution et le risque d'erreur augmente avec la taille des séries.

Pour toutes ces raisons, nous choisissons ainsi de baser notre architecture d'encodeur-décodeur sur le *Temporal Convolutional Autoencoder* (TCN-AE) [170], que nous avons présenté dans la section 2.1.3.4. En plus de satisfaire les critères précédents, le réseau a été utilisé avec succès pour la détection d'anomalies dans les ECG [170], témoignant de sa capacité à bien reconstruire les signaux de santé.

4.3.4 Apprentissage

Pour toutes les expériences, nous définissons notre encodeur comme un TCN-AE avec 6 piles de convolutions. Ce choix est déterminé expérimentalement selon le score sur le jeu de validation. Les facteurs de dilatation q_n sont respectivement 1,2,4,8,16,32, conformément à la publication originale [170]. Le décodeur est le même réseau, mais avec les facteurs de dilatation inversés. Nous optimisons les deux réseaux avec l'optimiseur Adam et un taux d'apprentissage $lr = 1e-4$. Nous fixons la taille du *batch* à 64 et entraînons l'encodeur pendant 20 *epochs*.

4.4 Résultats expérimentaux

4.4.1 Comparaisons avec les approches non paramétriques

Dans cette section, nous comparons les performances de notre méthode avec celles des algorithmes de compression par transformation, FFT, DCT et DWT.

Pour la compression via FFT, nous utilisons la fonction *resample* de la librairie *scipy*[3]. Cette fonction prend en entrée la série temporelle $x \in \mathbb{R}^N$ et le nouveau nombre d'échantillons voulu H. Elle calcule $x_{freq} = FFT(x) \in \mathbb{R}^N$, puis, si $H > N$ (sur-échantillonnage) applique le *padding* à x_{freq}, sinon (sous-échantillonnage) sélectionne les H coefficients les plus importants. Enfin, on obtient $x_{resampled} = iFFT(x_{freq})$. Dans notre cas, on utilise cette fonction 2 fois : on sous-échantillonne pour obtenir la représentation compressée $z = x_{resampled} \in \mathbb{R}^H$, puis on sur-échantillonne z pour obtenir $\hat{x}$. Nous nommons cette méthode *FFT-resampling* (FFT-r).

La compression via DCT est de nouveau réalisée via *scipy*[4]. On obtient directement $z = dct(x)$ de taille souhaitée en passant H en entrée de la fonction, puis on calcule $\hat{x} = idct(z)$.

Enfin, la compression via DWT est effectuée via la fonction *downcoef* de la librairie PyWavelets[5]. On choisit le niveau de décomposition (c'est-à-dire le nombre d'applications de la transformation) pour obtenir la taille H souhaitée. La reconstruction est faite via la fonction inverse *upcoef*, avec les mêmes paramètres.

Nous choisissons 3 facteurs de compression (CF) différents : 8, 32 et 64. Cela permet d'illustrer la capacité de notre architecture à bien compresser des séries à différents taux de compression avec pour seul changement la couche de *MaxPooling* finale (bien qu'il faille réapprendre le modèle pour chaque taux de compression donné). Nous utilisons dans un premier temps les métriques usuelles de système de compression : RMSE, SNR et QS. Nous présentons les résultats sur le jeu de données SleepEDF-78 dans le tableau 4.2.

Selon ces métriques, notre méthode est largement moins bonne que les approches non paramétriques, et en particulier inférieure au ré-échantillonnage via FFT (*FFT-r*),

3. https://docs.scipy.org/doc/scipy/index.html
4. https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.dct.html
5. https://pywavelets.readthedocs.io/en/latest/ref/dwt-discrete-wavelet-transform.html

Method	CF	RMSE ↓	SNR ↑	QS ↑
DCT	8	0.36	3.22	0.12
	32	0.44	1.50	0.35
	64	0.46	1.02	0.72
DWT (db1)	8	0.07	18.11	0.64
	32	0.10	14.46	1.69
	64	0.12	13.26	2.95
FFT-r	8	**0.06**	**19.16**	**0.73**
	32	**0.09**	**15.29**	**1.86**
	64	**0.11**	**13.69**	**3.09**
TCN-AE, DeepDTW-S	8	0.51	5.01	0.14
	32	0.84	0.77	0.35
	64	1.05	-0.9	0.58

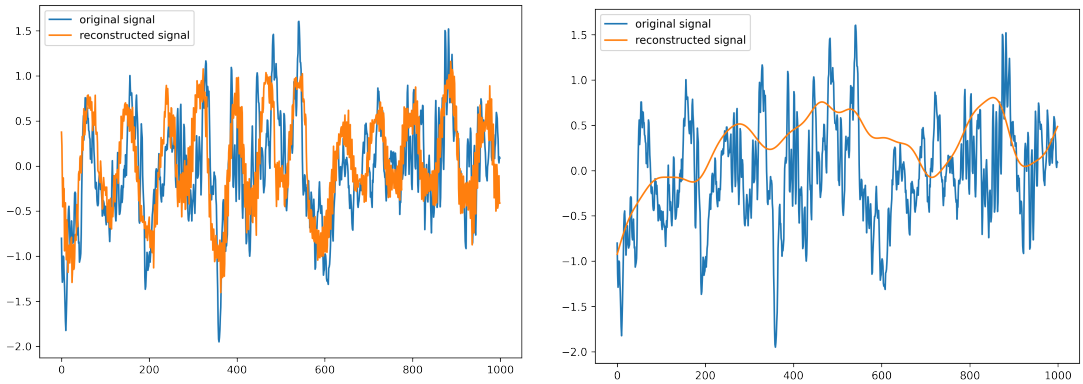
TABLE 4.2 – Résultats d'algorithmes non paramétriques et de notre méthode selon les métriques usuelles de compression, sur le jeu SleepEDF-78

qui est meilleur selon tous les critères. Néanmoins, nous rappelons que ces métriques se basent sur la distance euclidienne pour comparer les signaux reconstruits avec les signaux originaux, tandis que notre méthode (TCN-AE DeepDTW-S) est apprise en utilisant une approximation de DTW en fonction de coût.

Ces résultats vont de plus à l'encontre de l'intuition que nous avons en regardant les signaux reconstruits, affichés dans la figure 4.3. En effet, le signal reconstruit par la DCT semble très mauvais, ne suivant aucun des motifs caractéristiques des signaux de sommeil. Celui reconstruit par DWT suit davantage la tendance globale, mais la perte d'informations due à la reconstruction est visuellement significative. Seul le signal compressé par ré-échantillonnage par FFT s'approche convenablement du signal original. Enfin, celui reconstruit par notre méthode s'appuyant sur l'architecture siamoise d'approximation de DTW comme fonction de coût semble bien suivre la tendance du signal original, mais ajoute du bruit au signal reconstruit. Ainsi, dans la suite de ce chapitre, nous évaluons la compression via le score de classification obtenu sur les données compressées puis reconstruites.

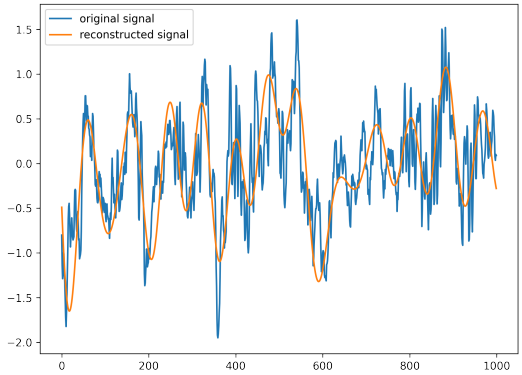
4.4.2 Classification sur les données reconstruites

Dans cette section, nous évaluons les performances de systèmes de compression en compressant puis décompressant des signaux. Nous apprenons ensuite un modèle de classification sur les données reconstruites. En inférence, la compression et la décompression sont deux opérations indépendantes, réalisées par deux modèles différents. Ainsi, plus le score obtenu par le modèle de classification sur les données reconstruites est haut, plus cela montre que le système de compression a réussi à garder les informations discriminantes des signaux.

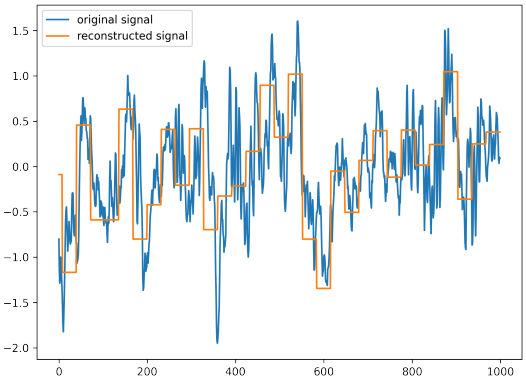


(a) TCN-AE Siamese

(b) DCT



(c) FFT-r



(d) DWT

FIGURE 4.3 – Comparaison entre les signaux originaux et reconstruits après compression ($CF = 32$) et décompression par notre méthode (TCN-AE avec DeepDTW Siamese comme fonction de coût), DCT, FFT-r et DWT

4.4.2.1 Apprentissage de la classification de sommeil

Nous spécialisons notre système de compression sur des données de sommeil. En effet, ce sont des séries longues (plusieurs heures de sommeil) échantillonnées à 100 Hz, créant ainsi de très longs signaux. Par exemple, un enregistrement de 8 heures de sommeil est représenté par un signal de 2 880 000 points par canal. Si la classification d'état du sommeil restreint la taille des signaux à 3000 points au minimum, ce n'est pas nécessairement le cas de toutes les tâches liées à ces données.

La tâche de classification du sommeil étant cependant très importante et représentée dans la littérature (comme nous l'avons développé dans la section 2.2), nous validons nos systèmes de compression via cette tâche.

Le protocole est le suivant :

— Apprentissage du modèle de compression sur le jeu d'entraînement D_{train} via la fonction de coût de reconstruction $\mathcal{L}$.

— Arrêt anticipé de l'apprentissage en suivant les valeurs de $\mathcal{L}$ sur le jeu de validation pour éviter le sur-apprentissage.

— Compression et décompression des jeux D_{train}, D_{val} et D_{test} pour obtenir les jeux

de données reconstruits $Drec_{train}$, $Drec_{val}$ et $Drec_{test}$

— Apprentissage du classifieur sur le jeu $Drec_{train}$, avec arrêt anticipé selon le score MF1 sur $Drec_{val}$

— Évaluation du classifieur sur le jeu $Drec_{test}$

Naturellement, dans le cas des méthodes ne nécessitant pas d'apprentissage (DCT, DWT et FFT-r), les 2 premières étapes sont ignorées.

En particulier, pour classifier le sommeil, nous utilisons le SorsNet, présenté dans la section 3.3.3.1. Cette fois, nous utilisons tout le réseau, comme proposé dans [159], avec un *dropout* de 30%. Nous utilisons la fonction de coût *Cross Entropy* et l'optimiseur *Adam* pour mettre à jour les poids de notre modèle avec un taux d'apprentissage de $1e^{-4}$. Le SorsNet est appris de zéro, avec un arrêt anticipé si le score MF1 ne s'améliore pas sur le jeu de validation pendant 10 *epochs* consécutives. L'apprentissage dure au maximum 100 *epochs*. Notre objectif est ici de comparer des méthodes de compression, pas d'améliorer l'état de l'art de la classification du sommeil. Ainsi, nous ne faisons pas de recherche d'hyperparamètres et utilisons le même classifieur et la même méthode d'apprentissage de classification pour tous les systèmes de compression.

4.4.2.2 Sur SleepEDF-78

Nous comparons ainsi les algorithmes DCT, FFT-r et DWT à notre méthode TCN-AE. Nous utilisons différentes fonctions de coût de reconstruction pour notre architecture : MSE, softDTW, *Physics Informed* (PI), LDPS, et les approches proposées dans le chapitre précédent : DeepDTW Siamoise (DeepDTW-S) et DeepDTW Directe (DeepDTW-D). Nous utilisons pour les approches LDPS et DeepDTW les poids des modèles appris sur le jeu de données SleepEDF-78 : les méthodes et les paramètres d'apprentissage sont décrits dans le chapitre précédent. Nous prenons soin de conserver les séparations du jeu de données D effectuées pour l'apprentissage de nos approximations de DTW pour éviter toute fuite de données.

Nous affichons les résultats dans le tableau 4.3. Pour référence, nous présentons le score obtenu par le SorsNet sur les données originales (non compressées). Avec un taux de compression de 8, notre architecture apprise via la DeepDTW Siamoise obtient le meilleur score de classification du sommeil, montrant que les données compressées ont conservé le plus d'informations. La tâche semble assez facile à ce taux, et toutes les méthodes sauf la DCT sont proches, entre 4 et 8 pp en dessous des résultats sur les données originales. Avec des taux plus élevés, la fonction de coût LDPS obtient des résultats légèrement meilleurs que notre approche Siamoise. De plus, l'écart devient plus significatif entre ces méthodes et les autres à mesure que la tâche se complexifie. Avec un taux de compression de 32, LDPS et notre méthode basée sur la DeepDTW-S sont autour de 3 pp supérieures à la méthode basée sur la DeepDTW-D et 7 pp supérieures aux autres méthodes. Enfin, avec un taux de compression de 64, la majorité des méthodes sont environ 11 pp inférieures à LDPS et notre approche siamoise, et 8 pp en dessous de notre approche directe. Nous remarquons aussi que les résultats des approches utilisant notre autoencodeur sans nos approximations de DTW sont quasiment identiques à ceux de FFT-r et DWT. Cela illustre que si la fonction de coût de reconstruction n'est pas adaptée, l'autoencodeur TCN-AE se comporte comme un

Balanced Accuracy / MF1 sur SleepEDF-78 par SorsNet			
Données originales : 0.73/0.73			
Modèle	CF = 8	CF = 32	CF = 64
DCT	0.59/0.59	0.49/0.48	0.42/0.41
DWT	0.66/0.66	0.53/0.51	0.45/0.41
FFT-r	0.65/0.66	0.54/0.52	0.44/0.43
TCN-AE MSE	0.65/0.65	0.53/0.50	0.44/0.42
TCN-AE PI	0.66/0.66	0.54/0.51	0.45/0.44
TCN-AE SoftDTW	0.65/0.65	0.54/0.52	0.45/0.42
TCN-AE LDPS	0.67/0.67	**0.61/0.61**	**0.58/0.58**
TCN-AE DeepDTW-D	0.65/0.65	0.57/0.56	0.53/0.52
TCN-AE DeepDTW-S	**0.68/0.68**	0.60/0.59	0.56/0.56

TABLE 4.3 – Score de classification d'état du sommeil sur SleepEDF-78. Les signaux sont compressés par différents modèles avec le facteur de compression CF puis décompressés avant d'être utilisées par un classifieur pour l'apprentissage et le test. La classification est apprise 3 fois et le score moyen est affiché.

autoencodeur linéaire.

4.4.2.3 Avec d'autres classifieurs

Nous avons montré qu'utiliser nos approximations de DTW en tant que fonction de coût améliore le score de classification de sommeil sur les données reconstruites, et donc permet de compresser les signaux en gardant plus d'informations.

Dans cette section, nous cherchons à évaluer l'universalité de cette amélioration en examinant son impact en utilisant d'autres classifieurs. Il est en effet possible que notre système soit biaisé, optimisant uniquement la reconstruction des signaux en fonction des caractéristiques spécifiques exploitées par SorsNet pour distinguer les états de sommeil. Pour éliminer ce biais potentiel, nous reproduisons l'expérience en substituant SorsNet par deux autres classifieurs de sommeil : 1D-CNN [55] et AttentionSleep [54].

Le 1D-CNN est une architecture simple et légère comprenant 3 blocs de convolution. Chacun contient une couche de convolution 1D, une couche de *BatchNorm*, une activation ReLU et une couche de *MaxPooling*. Cette architecture ne comprend aucun module spécifiquement construit pour représenter la temporalité des données. Nous le sélectionnons justement pour cette simplicité.

L'extracteur de caractéristiques du modèle AttentionSleep est aussi basé sur des couches de convolutions, mais l'encodeur temporel est composé de couches d'attention. Ce modèle sera décrit en détail dans le chapitre 5. Utiliser ce réseau nous permet d'étudier la capacité de nos systèmes de compression à conserver la temporalité des signaux.

Pour une comparaison équilibrée, nous utilisons la même fonction de coût pour tous

nos classifieurs : la *Cross Entropy*, et l'apprentissage se fait de manière identique à celui de SorsNet.

Balanced Accuracy / MF1 sur SleepEDF-78 par AttentionSleep			
Données originales : 0.72/0.72			
Reconstruction	CF = 8	CF = 32	CF = 64
DCT	0.58/0.59	0.50/0.50	0.44/0.44
DWT	0.66/0.65	0.55/0.53	0.46/0.44
FFT-r	0.65/0.65	0.54/0.52	0.44/0.42
TCN-AE MSE	0.60/0.60	0.56/0.54	0.46/0.44
TCN-AE SoftDTW	0.66/0.65	0.57/0.54	0.47/0.43
TCN-AE LDPS	0.66/0.66	**0.61/0.61**	**0.58/0.58**
TCN-AE DeepDTW-D	0.62/0.63	0.54/0.53	0.50/0.50
TCN-AE DeepDTW-S	**0.67/0.67**	0.60/0.59	0.56/0.55

TABLE 4.4 – Score de classification d'état du sommeil sur SleepEDF-78 par AttentionSleep sur signaux compressés puis reconstruits.

Balanced Accuracy / MF1 sur SleepEDF-78 par 1D-CNN			
Données originales : 0.71/0.71			
Reconstruction	CF = 8	CF = 32	CF = 64
DCT	0.58/0.58	0.46/0.46	0.41/0.41
DWT	0.64/0.63	0.53/0.51	0.39/0.38
FFT-r	0.63/0.64	0.52/0.51	0.43/0.41
TCN-AE MSE	0.58/0.57	0.52/0.51	0.44/0.41
TCN-AE SoftDTW	0.63/0.63	0.56/0.54	0.45/0.42
TCN-AE LDPS	0.65/0.65	**0.59/0.59**	**0.57/0.57**
TCN-AE DeepDTW-D	0.62/0.62	0.51/0.50	0.48/0.48
TCN-AE DeepDTW-S	**0.66/0.65**	0.59/0.58	0.54/0.54

TABLE 4.5 – Score de classification d'état du sommeil sur SleepEDF-78 par 1D-CNN sur signaux compressés puis reconstruits.

Nous présentons les résultats obtenus par le classifieur AttentionSleep dans le tableau 4.4 et ceux obtenus par le 1D-CNN dans le tableau 4.5. Les résultats sont globalement similaires à ceux obtenus via le SorsNet. Le modèle 1D-CNN obtient des résultats légèrement en dessous du SorsNet et de l'AttentionSleep, mais qui suivent la même tendance : notre méthode TCN-AE apprise via DeepDTW Siamoise permet au classifieur de mieux distinguer les classes des signaux reconstruits à un taux de compression de 8, alors que TCN-AE utilisant LDPS comme fonction de coût obtient les meilleurs résultats quand la tâche devient plus difficile.

4.4.2.4 Sur un autre jeu de données

Comme nous l'avons fait au chapitre précédent, nous étudions si les capacités de compression et de reconstruction de notre modèle peuvent s'étendre à d'autres jeux de

Reconstruction	Balanced Accuracy	MF1
Données originales	0.75	0.76
MSE	0.56	0.56
SoftDTW	0.56	0.55
LDPS	**0.61**	**0.61**
DeepDTW-D	0.53	0.54
DeepDTW-S	**0.61**	0.60

TABLE 4.6 – Score de classification d'état du sommeil (5 classes) sur SHHS par SorsNet sur signaux compressés puis reconstruits avec un facteur de compression de 32. Nos méthodes et LDPS sont apprises sur SleepEDF-78.

données. Nous apprenons et évaluons notre méthode de compression et nos modèles de la même manière que précédemment dans le chapitre sur le jeu de données SHHS.

Les résultats sur le jeu de données SHHS sont présentés dans le tableau 4.6. On rappelle ici que nos méthodes d'approximation de DTW (DeepDTW-D, DeepDTW-S), ainsi que LDPS, n'ont pas été apprises sur le jeu de données SHHS mais sur le jeu SleepEDF-78 et ne sont pas finetunées. Malgré cela, LDPS et notre méthode siamoise permettent quand même de mieux classifier les données reconstruites. L'approximation directe, qui généralise moins bien, est cependant impactée et obtient de moins bons résultats que la MSE et la SoftDTW.

4.5 Conclusion

La popularisation de dispositifs médicaux portables crée un volume significatif de séries temporelles qu'il faut déplacer et stocker. Compresser les séries temporelles permet de significativement faciliter ces tâches en réduisant la taille des séries. Cependant, il est nécessaire que l'information permettant de réaliser la tâche associée aux séries soit conservée pendant la compression et la décompression.

Nous avons présenté dans ce chapitre une méthode permettant de compresser des séries temporelles en s'assurant de minimiser les pertes de performances d'un classifieur sur les données reconstruites. En s'inspirant des travaux connexes de la littérature, nous avons appris la tâche de compression via un autoencodeur, composé d'un encodeur basé sur des convolutions 1D, et d'un décodeur basé sur des convolutions transposées. Cela nous permet d'obtenir un facteur de compression haut et paramétrable. Nous avons remplacé la MSE, usuellement utilisée comme fonction de coût de reconstruction, par nos approximations de DTW. Nous avons ainsi obtenu une métrique d'erreur plus adaptée aux séries temporelles.

Nous avons appris la compression sur le jeu de données de classification du sommeil SleepEDF-78. Nous avons montré que notre proposition permet d'obtenir un score de classification sur les données reconstruites significativement plus élevé que les méthodes de compression usuelles (DCT, DWT et FFT). Nous avons aussi montré qu'utiliser

notre approximation siamoise de DTW en tant que fonction de coût permet d'obtenir un meilleur score de classification qu'en utilisant la MSE ou softDTW, et des performances quasi identiques à l'utilisation de LDPS. Enfin, nous avons illustré que ces résultats sont indépendants du choix du classifieur, et que nous pouvons les étendre à des jeux de données similaires.

Chapitre 5

Projet PANDORE-IA : détection de l'impact du COVID long sur le sommeil

Sommaire

5.1 Introduction 122
5.2 Projet PANDORE-IA 123
5.2.1 Acteurs du projet 123
5.2.2 Objectifs scientifiques du projet 124
5.3 Etat de l'art : détection du COVID long 125
5.4 Jeu de données COVISLEEP 126
5.4.1 Acquisition du jeu de données 126
5.4.2 Statistiques 128
5.4.3 Préparation du jeu de données 128
5.5 Classification du sommeil 129
5.5.1 Modèles comparés 130
5.5.2 Résultats 134
5.6 Identification du COVID long 137
5.6.1 Statistiques de sommeil 137
5.6.2 Classification 137
5.7 Conclusion du chapitre 143

Projet PANDORE-IA : détection de l'impact du COVID long sur le sommeil

5.1 Introduction

L'une des particularités du COVID-19 est la présence de symptômes plusieurs mois après l'infection : on parle alors de COVID long [48], condition concernant au moins 10 % des personnes infectées par le virus SARS-CoV-2, soit 65 millions de personnes dans le monde [13]. Les symptômes sont extrêmement variés, notamment douleur dans la poitrine et palpitations, toux, nausée, perte de mémoire et insomnie chronique [47]. L'insomnie chronique est la perturbation du sommeil la plus commune chez les personnes atteintes de COVID long [112]. Afin de contribuer à la recherche d'un traitement pour les patients concernés, il est nécessaire de mieux comprendre l'impact du COVID long sur le sommeil. En particulier, il est intéressant de déterminer si l'insomnie chronique liée au COVID diffère de l'insomnie chronique usuelle, c'est-à-dire si elle résulte de modifications du sommeil dues au COVID long, ou si elle est causée par le choc de la phase aiguë de la maladie. En particulier, l'étude de polysomnographies, enregistrements de l'activité cérébrale pendant le sommeil, pourrait permettre d'identifier des marques du COVID long. Cependant, une étude préliminaire utilisant des statistiques extraites de polysomnographies n'a pas suffi à séparer une population de personnes souffrant d'insomnie chronique non liée au COVID d'une population de personnes souffrant d'insomnie chronique après infection du COVID [140].

Nous proposons donc, dans le cadre du projet PANDORE-IA, partenariat entre l'entreprise Saagie [1] et l'unité de recherche VIFASOM [2], d'utiliser des modèles de classification de sommeil pour distinguer les enregistrements de sommeil des individus insomniaques qui sont affectés par le COVID de ceux qui ne le sont pas.

Pour ce faire, et en se basant sur [140], nous avons créé, à l'aide de l'hôpital Hôtel Dieu, un ensemble de données constitué de polysomnographies réalisées sur des patients souffrant d'insomnie induite par le COVID long. Pour chaque polysomnographie effectuée sur des patients atteints de COVID long, nous incluons les signaux de deux patients contrôles souffrant d'insomnie chronique, mais pas de COVID long. Cette démarche vise à déterminer si l'insomnie induite par le COVID long est différente d'autres formes d'insomnie chronique. Nous mettons ce jeu de données à la disposition de la communauté. C'est à notre connaissance le premier jeu de données de polysomnographies contenant des enregistrements de sommeil de personnes souffrant de COVID long.

Ensuite, parce que les caractéristiques spécifiques du COVID long pourraient varier selon le stade du sommeil, nous étiquetons les signaux de polysomnographie en suivant les recommandations de l'AASM [74]. Ceci nous permet de réaliser une classification

1. https://www.saagie.com/fr/
2. https://www.frcneurodon.org/unite-vifasom/

automatique des stades de sommeil sur cet ensemble de données, puis de comparer divers algorithmes à l'état de l'art sur ce jeu. Le fait que chaque patient souffre d'insomnie chronique avec divers symptômes rend la tâche de classification automatique des stades de sommeil difficile, et donc intéressante pour la communauté.

Enfin, nous abordons le problème de la détection de l'impact du COVID long sur le sommeil. Nous comparons la macro-architecture[3] du sommeil des deux populations, y compris le temps passé éveillé après le début du sommeil (*Wake After Sleep Onset*), l'efficience du sommeil (*Sleep Efficiency*) et le temps passé dans divers stades de sommeil. Nous montrons que ces paramètres sont insuffisants pour discriminer les patients souffrant d'insomnie induite par le COVID long de ceux atteints d'insomnie chronique. Nous apprenons également des modèles de classification pour prédire si des fenêtres de polysomnographies proviennent de patients infectés par le COVID long ou non. Nous utilisons des modèles spécifiquement conçus pour la classification des stades de sommeil et découvrons que les stades de sommeil N3 et REM semblent les plus affectés par le COVID long. Cependant, ces résultats restent limités du fait de la forte variance entre les patients.

Dans ce chapitre, nous présentons le projet PANDORE-IA, en nous intéressant au contexte, acteurs et objectifs du projet. Nous faisons un bref état de l'art de l'étude de l'impact du COVID long sur le sommeil. Ensuite, nous expliquons le processus de collecte du jeu de données extrait du projet PANDORE-IA que nous mettons à disposition de la communauté. Nous nommons le jeu de données COVISLEEP. Nous comparons différentes approches de classification du sommeil à l'état de l'art afin de déterminer la plus adaptée à notre jeu de données. Enfin, nous explorons la détection du COVID long dans des enregistrements de sommeil, d'abord à l'aide de statistiques descriptives, puis en appliquant une classification binaire.

5.2 Projet PANDORE-IA

Le projet PANDORE-IA, collaboration entre la société Saagie, le laboratoire VIFASOM et l'institut IRBA, vise à développer des méthodes d'intelligence artificielle pour analyser des données physiologiques (EEG) dans le contexte de la recherche biomédicale. L'équipe mixte de chercheurs, ingénieurs et médecins concevra des modules logiciels (plateforme Saagie) et modèles prédictifs (algorithmes de Machine Learning) pour répondre aux besoins d'experts de la santé (VIFASOM). Nous présentons les différents acteurs du projet dans la suite.

5.2.1 Acteurs du projet

Saagie est une société se spécialisant dans la facilitation de la transition numérique en aidant les entreprises à surmonter les obstacles techniques associés au Big Data. En plus de fournir des services à une variété de clients, tant français qu'internationaux (comme Vente privée, Johnson & Johnson, Icade, Caisse d'Epargne Normandie, Ferrero, Bouygues Energies Services, Matmut, etc.), Saagie a également mis en place sa

3. c'est-à-dire des statistiques liées aux états du sommeil et non aux enregistrements des capteurs

propre plateforme Big Data. Cette solution clé en main, complète et axée sur des applications spécifiques, permet aux entreprises de tous les secteurs de maximiser la valeur de leurs données et de créer de nouvelles opportunités de croissance [19]. L'équipe de Saagie est multidisciplinaire et comprend des data scientists, des développeurs, des architectes de données et des experts dans divers domaines, en plus d'un département de recherche. Ce dernier est axé sur trois domaines principaux : l'apprentissage faiblement supervisé, l'apprentissage multimodal et l'explicabilité des modèles. Il vise également à intégrer de nouvelles technologies liées à l'apprentissage profond dans le produit Saagie et à accroître la visibilité de l'entreprise par le biais de compétitions.

L'Institut de Recherche Biomédicale des Armées (IRBA), situé à Brétigny-sur-Orge en Île-de-France, est un centre de recherche spécialisé du Service de Santé des Armées. Il est dédié à la recherche médicale, avec un focus particulier sur les conditions et les environnements dans lesquels les forces armées opèrent, y compris les risques associés au nucléaire, à la radiologie, à la biologie et à la chimie. Fondé en 2009, l'IRBA a combiné les ressources de quatre établissements de recherche dans le but d'améliorer la santé des militaires. L'institut a une double mission. D'une part, il répond aux besoins spécifiques identifiés par les hauts commandements militaires et le service de santé des armées pour protéger le personnel militaire. D'autre part, il travaille à anticiper les futurs besoins et à renforcer les mesures médicales pour faire face à diverses contraintes opérationnelles. La combinaison de l'expertise scientifique et de la connaissance du contexte militaire distingue l'IRBA et lui donne une place unique dans le domaine de la recherche en France.

Fondée en 2013 à l'Université Paris Descartes, l'équipe de recherche EA 7330 VIFASOM (Vigilance Fatigue Sommeil et santé publique) est codirigée par le Professeur Damien LEGER et Mounir CHENNAOUI de l'Institut de Recherche Biomédicale des Armées. Elle compte environ une vingtaine de chercheurs de l'IRBA et de l'université, spécialisés dans divers aspects liés au sommeil, comme l'épidémiologie, les impacts de la privation de sommeil sur la vigilance, la performance et le risque de blessure, entre autres.

Le projet est financé dans le cadre du dispositif RAPID par la Direction Générale des Armées (DGA).

Dans la suite, nous rentrons dans le détail des objectifs scientifiques du projet.

5.2.2 Objectifs scientifiques du projet

Le projet PANDORE-IA est axé sur trois tâches principales :

— La détection de fatigue durant la conduite. En utilisant des EEGs obtenus via bandeau Dreem pendant un test de conduite, on cherche à prédire la perte de vigilance des conducteurs. On identifie une perte de vigilance selon les erreurs commises par le volontaire durant le test de conduite (sortie de route).

— La détection de l'impact de la caféine sur les performances de conduite. Des volontaires participent à deux tests de conduite : le premier en soirée et le deuxième après une nuit avec restriction totale de sommeil. Une partie des vo-

lontaires consomme de la caféine avant le deuxième test, tandis que l'autre partie consomme un placébo. L'objectif est alors d'illustrer l'impact de la caféine sur les performances de conduite sur des volontaires en manque de sommeil.

— La détection de l'impact du COVID long sur le sommeil de patients insomniaques. L'objectif de la tâche est de comparer, grâce à des modèles de *Machine Learning*, les polysomnographies de patients insomniaques souffrant de COVID long (dits sujets) et de patients insomniaques non infectés par le virus (dits contrôles) pour identifier des différences sur les signaux de sommeil des patients affectés par le COVID long, et ainsi aider à développer un traitement adapté.

Dans ce manuscrit, nous présenterons les travaux réalisés sur la troisième tâche : la détection de l'impact du COVID long sur le sommeil. Celle-ci est séparée en deux phases principales : l'obtention et la préparation du jeu de données, et l'étude de l'impact du COVID long.

Nous commençons par présenter un état de l'art de la détection du COVID.

5.3 Etat de l'art : détection du COVID long

Le COVID long (parfois appelé séquelles post-aiguës du COVID-19) est une affection multi-systémique comprenant souvent des symptômes graves qui suivent une infection par le syndrome respiratoire aigu coronavirus 2 (SARS-CoV-2) [48], touchant au moins 65 millions de personnes dans le monde [13]. Le COVID long est défini par World Health Organization (WHO) comme la poursuite ou l'apparition de nouveaux symptômes 3 mois après l'infection initiale, ces symptômes persistant pendant au moins 2 mois sans autre explication [158]. Les symptômes du COVID long sont nombreux et disparates [47]. En particulier, l'étude internationale sur le sommeil et le COVID (ICOSS-II) a découvert que les symptômes d'insomnie, la fatigue et la somnolence diurne excessive étaient parmi les plaintes les plus courantes liées au COVID long [108]. Selon [112], 25% des patients se plaignent de troubles de sommeil, dont 22.2% d'insomnies.

L'insomnie est définie comme une difficulté à initier ou à maintenir le sommeil, ou un sommeil non réparateur, et comme causant une détresse cliniquement significative ou une altération du fonctionnement social et professionnel [133]. L'insomnie est dite chronique quand elle se manifeste au moins 3 fois par semaine et depuis au moins 3 mois [81]. Bien que ce soit un dysfonctionnement du sommeil, l'insomnie chronique n'est usuellement pas diagnostiquée par polysomnographie [97]. À notre connaissance, seul le jeu de données *Sleep-Telemetry* (sous-partie du jeu SleepEDF) est constitué de polysomnographies des nuits de sommeil de patients insomniaques (voir 2.2.1.2).

Du fait du manque de polysomnographies de personnes insomniaques et pour étudier l'insomnie chronique liée au COVID long, [140] ont recueilli les polysomnographies de 17 patients souffrant d'insomnie après infection du COVID, et 34 patients souffrant d'insomnie sans infection du COVID. L'objectif était d'essayer d'illustrer une différence entre les deux populations via des statistiques de sommeil. Cependant, seul l'Apnea Hypopnea Index (AHI, qui compte le nombre d'événements d'apnée et d'hypopnée

pour en quantifier la sévérité) permet de séparer les deux populations au sens du test de Student. Enfin, le manque de patients et le fait que l'information d'apnée n'était pas présente chez tous les patients rend une conclusion difficile.

Dans le cadre du projet PANDORE-IA, nous proposons donc, en nous basant sur les travaux de [140], de créer un jeu de données de polysomnographies de personnes souffrant d'insomnie chronique liée au COVID, et de personnes insomniaques n'ayant pas été infectées. Étant donné l'insuffisance des résultats obtenus avec l'approche qui se base sur l'analyse de la macro-structure du sommeil, nous proposons d'utiliser des modèles de classification du sommeil, expliqués en profondeur dans la section 2.2. Notre objectif est d'utiliser ces modèles pour distinguer les données de sommeil issues de personnes atteintes de COVID long de celles qui ne le sont pas.

Malgré la récence de la pandémie due au COVID-19, l'impact à long terme de l'infection sur le sommeil a été largement étudié par la communauté scientifique. Cependant, la compréhension des mécanismes engendrés par le COVID long, notamment sur l'insomnie chronique, reste incomplète. Bien que des polysomnographies de personnes souffrant d'insomnie chronique due au COVID aient été réalisées, le manque de volume du jeu de données, ainsi que la nature généralement privée des données de santé empêche la progression de la recherche.

Nous proposons donc d'acquérir et de publier un jeu de données de polysomnographies de personnes insomniaques suite à une contamination par le virus, accompagné de polysomnographies de personnes insomniaques similaires, mais non infectées.

Dans la suite de ce chapitre, nous présentons le jeu de données COVISLEEP que nous avons collecté et que nous mettons à disposition de la communauté.

5.4 Jeu de données COVISLEEP

Notre objectif est de déterminer si l'insomnie chronique causée par le COVID long présente des caractéristiques différentes de l'insomnie chronique usuelle. L'insomnie impactant le sommeil, les experts du centre de sommeil VIFASOM ont choisi de réaliser des polysomnographies de nuit de sommeil de patients insomniaques indiquant souffrir de symptômes de COVID long (sujets) et de patients insomniaques n'ayant pas été infectés par le COVID (contrôles). Pour ce faire, un ensemble de capteurs Nox [4] perturbant peu le sommeil des patients ont été utilisés.

5.4.1 Acquisition du jeu de données

L'ensemble des données a été rassemblé à partir des services des maladies infectieuses et du sommeil de l'Hôtel-Dieu à Paris, en France. Le détail des méthodes d'obtention des données d'un point de vue médical est développé dans [140], et sera synthétisé dans ce chapitre.

Les patients atteints de COVID long ont été choisis parce qu'ils se plaignaient d'in-

4. https://noxmedical.com/

somnie chronique suite à leur infection par le COVID-19. Un spécialiste du sommeil de l'hopital Hôtel-Dieu a confirmé le diagnostic d'insomnie chronique en se basant sur les critères de classification ICSD-3 et DSM-5 [151]. Les patients inclus étaient des adultes qui ne souffraient pas d'autres troubles du sommeil et ne prenaient pas de médicaments affectant le système nerveux. Les polysomnographies ont été effectuées entre 6 et 18 mois après une infection aiguë par le COVID-19. La date d'infection, déclarée par les patients, est renseignée dans le jeu de données pour 24 des 31 patients souffrant de COVID long. Pour les personnes atteintes d'insomnie chronique, la sélection a été effectuée consécutivement dans le même service de sommeil et pendant la même période. Pour chaque patient atteint de COVID étudié, deux sujets contrôles, non infectés, ont été inclus, appariés en fonction de l'âge, du sexe et de l'indice de masse corporelle (IMC). Tous les sujets contrôles étaient également diagnostiqués par un médecin comme souffrant d'insomnie chronique, mais n'avaient pas signalé d'infection aiguë par le COVID-19 ou de symptômes de COVID long.

Dans chaque polysomnographie réalisée, jusqu'à 80 canaux de données ont été enregistrés. Cependant, notre étude se concentre spécifiquement sur l'impact du COVID long sur le cerveau pendant le sommeil. Par conséquent, nous avons retenu 8 canaux pour notre analyse. Parmi eux, 6 canaux EEG (C3-M2, C4-M1, F3-M2, F4-M1, O1-M2, O2-M1) correspondent à une différence de potentiel entre les zones centrales (C), frontales (F) et occipitales (O) du cerveau par rapport à des références mastoïdiennes (M). Le numéro associé indique le côté du cerveau considéré. Nous avons également conservé 2 canaux EOG : E1-M2, E2-M1. Tous ces canaux sont initialement échantillonnés à une fréquence de 200 Hz.

La classification manuelle des stades de sommeil a été effectuée par un médecin spécialisé en troubles du sommeil et un technicien du sommeil ayant tous deux plus de 5 ans d'expérience. Cette classification a été faite conformément aux recommandations de l'AASM [74] et en utilisant les données des canaux EEG et EOG précédemment mentionnés. Les événements respiratoires ont également été pris en compte. Ainsi, un état de sommeil parmi « Éveil », « N1 », « N2 », « N3 », « N4 »et « REM »est associé à chaque fenêtre de 30 secondes de signal.

Diverses métadonnées concernant les patients ont été recueillies et synthétisées : l'âge, la taille, la masse, l'IMC, ainsi que la date de la polysomnographie. Un diagnostic du type d'insomnie (parmi « insomnie maintien », « insomnie endormissement », « insomnie endormissement et maintien », « insomnie maintien et réveil précoce », « insomnie retard de phase ») est également fourni pour tous les patients.

Des épisodes d'apnée (une diminution du flux respiratoire de 90 % pendant plus de 10 secondes [106]) et d'hypopnée (une diminution du flux respiratoire de 30 % à 90 % associée à un éveil ou à une désaturation en oxygène de plus de 3 % et pendant plus de 10 secondes [106]) ont également été enregistrés et inclus dans l'ensemble de données.

5.4.2 Statistiques

Pour synthétiser le jeu de données, nous calculons diverses statistiques résumant notre jeu de données suivant [67], et les présentons dans le tableau 5.1. Ces statistiques sont calculées selon les annotations expertes de sommeil via les algorithmes de la librairie Yet Another Sleep Algorithm (YASA [5]), qui permet de facilement charger et analyser les enregistrements de sommeil en Python, et qui implémente des algorithmes de classification automatique du sommeil. Les statistiques sont calculées par patient, sans pondérer selon la durée du signal. Cela implique que bien que la durée de l'acquisition des signaux ne soit pas la même pour tous les patients, nous présentons la moyenne et l'écart type des statistiques calculées sur chaque patient. Suivant diverses contributions de classification du sommeil présentées dans le chapitre 2 (par exemple SorsNet [159], DeepSleepNet [165], Sleep-Linear [174]), nous fusionnons les états N3 et N4 dans tout ce chapitre.

Age	42.97 ± 11.75
H/F	15 (5 covid) / 78 (26 covid)
Indice de Masse Corporelle	23.45 ± 4.43
Time in Bed (min)	465.67 ± 71.3
Sleep Period Time (min)	422.01 ± 64.88
Sleep Time (min)	341.27 ± 62.73
Sleep Efficiency (%)	73.95 ± 12.99
Sleep Onset Latency (min)	32.49 ± 32.08
Wake After Sleep Onset (min)	80.73 ± 62.85
Éveil (%)	26.05 ± 12.99
N1 (%)	4.01 ± 3.45
N2 (%)	38.05 ± 9.29
N3 (%)	18.48 ± 7.82
REM (%)	13.41 ± 6.71

TABLE 5.1 – Statistiques du jeu de données COVISLEEP. *Time in Bed* représente la durée totale de l'expérimentation, *Sleep Period Time* représente la durée entre la première et la dernière phase de sommeil de la nuit. *Sleep Time* est le temps réellement passé à dormir (états N1, N2, N3 et REM), la *Sleep Efficiency* est le ratio entre la durée de l'expérience et *Sleep Time*, *Sleep Onset Latency* indique la durée avant le premier endormissement, *Wake After Sleep Onset* indique la durée d'éveil après le premier endormissement.

5.4.3 Préparation du jeu de données

Dans cette section, nous décrivons les prétraitements apportés au jeu de données. Suivant les pratiques usuelles en classification du sommeil ([93], [67], [174]), nous ré-échantillonnons les signaux à 100 Hz. En conséquence, nous obtenons pour chaque patient une matrice de données $X_{pat} \in \mathbb{R}^{N_{wp} \times 3000 \times 8}$, où N_{wp} représente le nombre de fenêtres extraites de l'enregistrement d'un patient donné, 3000 indique la taille des

5. https://raphaelvallat.com/yasa/build/html/generated/yasa.sleep_statistics.html

ID fold	ID du trio de patient
0	2,6,7,12,17,29
1	8,9,10,15,18,31
2	5,19,24,25,27,28
3	1,4,14,20,21,22
4	3,11,13,16,23,26,30

TABLE 5.2 – Distribution de trios de patients dans les *folds*.

signaux et 8 le nombre de canaux (6 EEG et 2 EOG). De plus, nous synchronisons les enregistrements de polysomnographie avec les labels associés, car pour la plupart des patients, les premières fenêtres de sommeil n'ont pas été annotées. De même, nous excluons toute fenêtre associée à des étiquettes manquantes, ou ne faisant pas partie des classes de sommeil.

Après avoir regroupé les données de tous les patients, on obtient le jeu de données complet $X \in \mathbb{R}^{86391\times3000\times8}$: le jeu de données est donc constitué de 86391 fenêtres de 30 secondes comportant 8 canaux échantillonnés à 100 Hz.

Pendant l'acquisition de l'ensemble de données, les patients ont été soigneusement choisis pour établir un ratio de 2 contre 1 entre les contrôles et les sujets : on obtient ainsi 31 trios de patients. Afin de faciliter l'évaluation de nos méthodologies, nous divisons aléatoirement ces groupes en cinq *folds* séparés, chacun comprenant soit 6, soit 7 trios de patients. Cela garantit que le modèle ne puisse jamais apprendre et être évalué sur le même patient, ou sur un patient ayant des métadonnées similaires. Pour faciliter la reproduction de nos expériences, nous présentons les *folds* dans le tableau 5.2.

Une fois le jeu de données préparé, nous choisissons et comparons des algorithmes de classification du sommeil.

5.5 Classification du sommeil

Dans cette section, nous évaluons les performances de différents modèles de classification du sommeil sur le jeu de données COVISLEEP. En effet, puisque nous supposons que l'impact du COVID long est identifiable dans les signaux de sommeil, nous cherchons à déterminer le modèle qui classifie le mieux les états du sommeil, pour l'utiliser ensuite pour tenter de détecter les marques de COVID. Ainsi, nous sélectionnons différents modèles proposés dans la littérature selon leurs performances sur d'autres jeux de sommeil (SleepEDF [83] et SHHS [186], présentés dans la section 2.2.1) et selon leur architecture, dans le but d'obtenir un panel des approches utilisées pour classifier le sommeil. Cela permet de déterminer l'architecture la plus adaptée au jeu COVISLEEP. Nous présentons dans la suite de cette section les approches sélectionnées.

5.5.1 Modèles comparés

Il serait difficile de tester toutes les approches proposées pour la classification du sommeil du fait de leur nombre significatif (nous en présentons une dizaine dans la section 2.3, et cela reste loin d'être exhaustif). Ainsi, nous choisissons d'utiliser des approches dont les performances s'approchent des meilleures enregistrées à ce jour, utilisant différentes méthodes d'extraction des caractéristiques. Nous choisissons donc Sleep-Linear, dont les caractéristiques sont définies à la main, ainsi que des méthodes basées sur des réseaux de neurones utilisant des réseaux de convolution (1D-CNN [55], DeepSleepNet [165]), des réseaux à attention (AttentionSleep [54]) ainsi qu'un réseau utilisant une combinaison pyramidale de méthode d'extraction de caractéristiques (SleepyCo [93]). Ces approches sont détaillées dans la suite.

La méthode **Sleep-Linear**, proposée par [174], génère des caractéristiques à partir des signaux EEG, EOG et EMG. Initialement, un ensemble complet de 131 caractéristiques est créé, couvrant à la fois les domaines temporel et fréquentiel. Dans le domaine temporel, des mesures statistiques telles que l'écart-type, l'écart interquartile et le nombre de passages par zéro sont inclus. En complément de celles-ci, des attributs du domaine fréquentiel tels que les ratios de puissance spectrale alpha à theta et les statistiques spectrales basées sur la transformée de Fourier sont également intégrés dans l'ensemble de caractéristiques.

Ces caractéristiques sont calculées pour chaque fenêtre de 30 secondes $x^{(t)} \in \mathbb{R}^{3000,8}$. Nous rappelons que l'état de sommeil $y^{(t)}$ n'étant pas indépendant de l'état précédant $y^{(t-1)}$ (illustré dans la section 2.2.3), utiliser l'information des fenêtres passées et futures peut améliorer les performances du modèle. Ainsi, [174] crée aussi des caractéristiques pour les deux fenêtres précédentes ($x^{(t-2)}$ et $x^{(t-1)}$) ainsi que les deux suivantes ($x^{(t+1)}$ et $x^{(t+2)}$). Enfin, des caractéristiques sont créées pour la concaténation de $\{x^{(t-1)}, x^{(t)}\}$, $\{x^{(t)}, x^{(t+1)}\}$ et $\{x^{(t-1)}, x^{(t)}, x^{(t+1)}\}$. On résume ce processus dans la figure 5.1. On obtient ainsi un vecteur de caractéristiques $X_{feat} \in \mathbb{R}^{N_r \times 1024}$ avec N_s le nombre de fenêtres d'un individu et $N_r = N_s - 4$ (les deux premières et deux dernières fenêtres sont ignorées pour permettre le calcul).

Nous apprenons Sleep-Linear via le code fourni par les auteurs de la méthode [6], sans modifier les paramètres d'aucun des algorithmes de classification.

Suivant [174], nous utilisons un classificateur linéaire SVM appris par *Stochastic Gradient Descent* (SGD [7]), un Random Forest (RF [8]) et un classificateur basé sur des arbres boostés par gradient (Catboost [9]) pour classifier et fournir des prédictions interprétables (au sens où nous pouvons calculer l'importance des caractéristiques).

1D-CNN représente l'utilisation d'un modèle composé uniquement de blocs de convolution 1D (couche de convolution, *batch normalization*, activation ReLU et Max Poo-

6. https://github.com/predict-idlab/sleep-linear
7. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
9. https://catboost.ai/

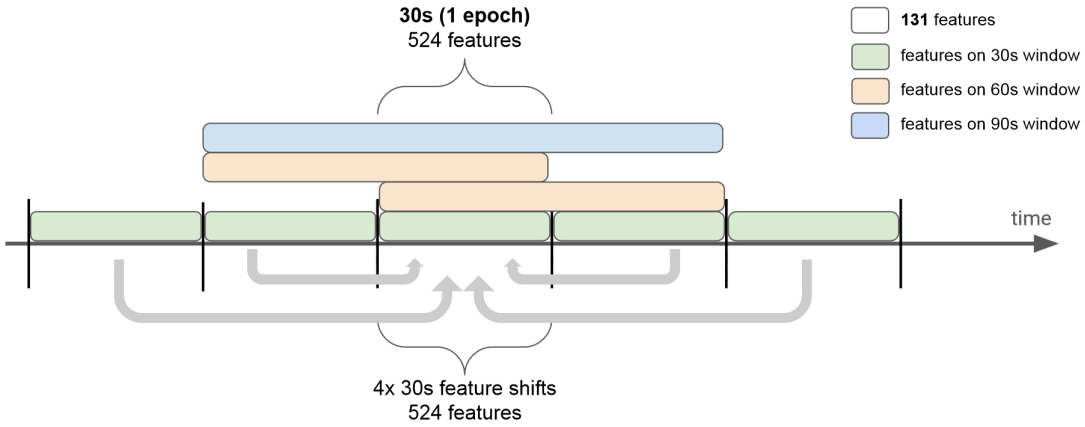


FIGURE 5.1 – Extraction de caractéristiques selon [174]. $x^{(t)}$ est la fenêtre centrale. On ajoute les caractéristiques calculées sur les 2 fenêtres précédentes et suivantes, ainsi que sur la concaténation des fenêtres adjacentes.

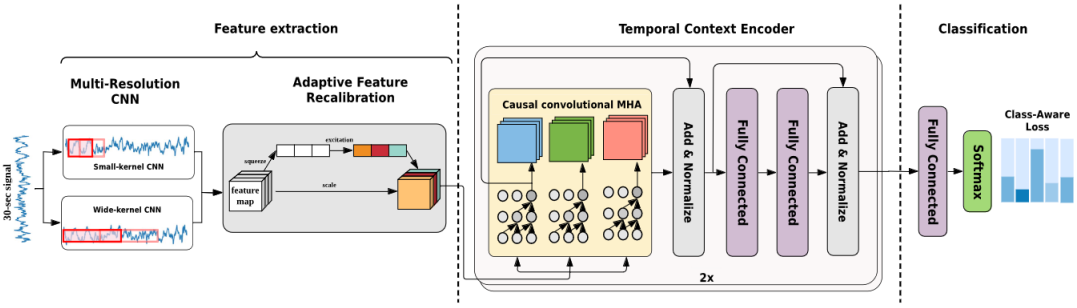


FIGURE 5.2 – Architecture globale du modèle AttentionSleep. Extrait de [54]

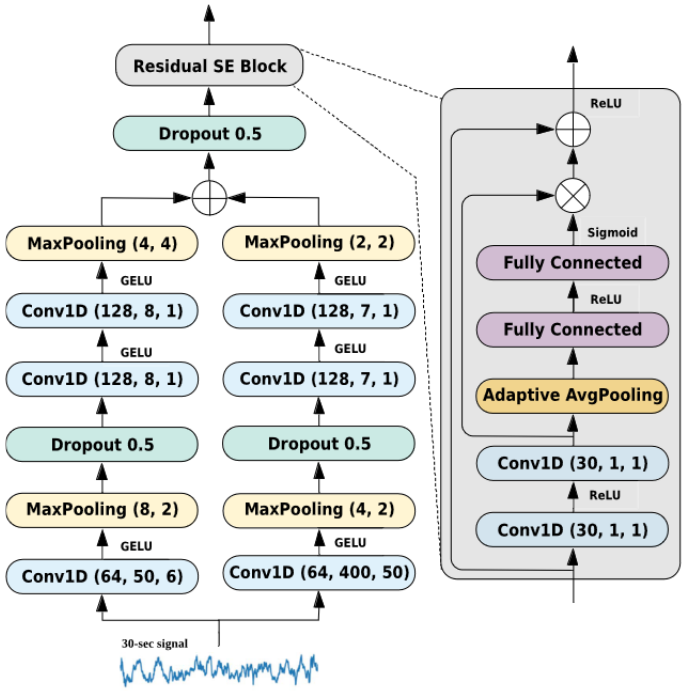


FIGURE 5.3 – Architecture de l'extracteur de caractéristiques d'AttentionSleep. Extrait de [54]. Conv1D (64,50,6) désigne une couche de convolution 1D avec 64 filtres, une taille de noyau de 50 et un *stride* de 6. GELU est une fonction d'activation introduite par [70].

ling 1D) jusqu'à une dernière couche dense servant à effectuer la classification. Différentes modifications du 1D-CNN ont été proposées du fait de ses bonnes performances en classification du sommeil [184], [188], [55]. Ici, nous utilisons le 1D-CNN comme *baseline* : nous choisissons donc la version proposée dans [56], simplement composée de 3 blocs de convolutions et d'une couche dense.

Le modèle **AttentionSleep** [54], dont l'architecture globale est illustrée dans la figure 5.2, est constitué de 3 éléments principaux : l'extracteur de caractéristiques, l'encodeur temporel et la couche de classification, comme la majorité des architectures modernes de classification du sommeil (décrites dans la section 2.2.3). L'extracteur de caractéristiques est détaillé dans la figure 5.3. Chaque signal $x \in \mathbb{R}^N$ est traité par deux branches composées de couches convolutionnelles avec des tailles de signaux et des *strides* différents, permettant de représenter les signaux à différentes échelles. Les deux représentations obtenues sont concaténées puis un bloc résiduel, composé de convolutions 1D et de couches denses, termine l'extraction de caractéristiques. Ensuite, l'encodeur temporel, composé d'une couche d'auto-attention multi-tête causale (détaillée dans la section 2.1.3.3) liée à des couches denses via des *skip-connexion* (voir 2.1.3.3) termine la représentation du signal. La classification est simplement effectuée via une couche dense.

Nous rappelons que le **DeepSleepNet** [165], que nous illustrons dans la figure 5.4, est constitué des mêmes éléments que l'AttentionSleep : un extracteur de caractéristiques, un encodeur temporel et une couche de classification. L'extracteur de caractéris-

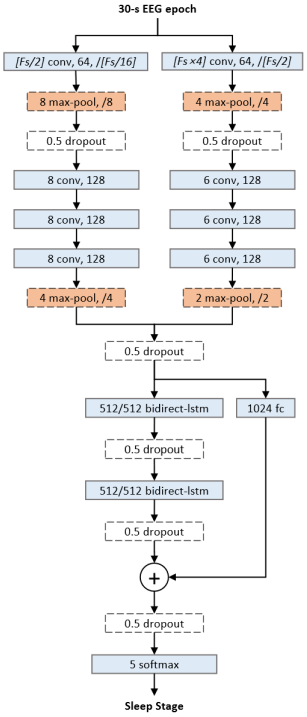


FIGURE 5.4 – Rappel de l'architecture du DeepSleepNet [165]. Le modèle est décrit dans le détail dans la section 2.2.3.

tiques est ici formé par deux branches de blocs de convolutions contenant convolution 1D, MaxPooling et dropout. Les nombres de filtres et *strides* des convolutions sont différents pour obtenir deux représentations des signaux d'entrée, qui sont ensuite concaténées. L'encodeur temporel est constitué d'une suite de deux couches de LSTM bidirectionnelles, avant qu'une couche dense effectue la classification.

Nous utilisons l'implémentation [10] fournie par [56] pour l'apprentissage du 1D-CNN, AttentionSleep et DeepSleepNet. Les trois modèles sont appris durant 40 epochs sur chaque *fold* via la fonction de coût *Cross Entropy*.

L'architecture du modèle **SleepyCo** [93] est représentée dans la figure 5.5. Le modèle reçoit L fenêtres $x^{(L)} \in \mathbb{R}^N$ consécutives et prédit les $\hat{y}^{(L)}$ états du sommeil correspondants. SleepyCo comporte trois composants majeurs : un *backbone network*, des *lateral connection*, et un module de classification *classifier network*. Le *backbone network* extrait des séquences de caractéristiques des signaux EEG bruts à plusieurs échelles temporelles. On obtient des caractéristiques selon :

$$\{C_3^{(L)}, C_4^{(L)}, C_5^{(L)}\} = f(x^{(L)}; \theta_f) \tag{5.1}$$

avec $C_i^{(L)}$ la sortie du i-ème bloc convolutionnel, f le *backbone network* et θ_f les paramètres du réseau. Chaque bloc convolutionnel est composé de répétitions de couches de convolution, *batch normalization*, et de la PReLU [68] comme fonction d'activation. Les *lateral connection* ont pour but de transformer les séquences de caractéristiques extraites par le *backbone network*, qui peuvent avoir différentes dimensions en une

10. https://github.com/emadeldeen24/eval_ssl_ssc

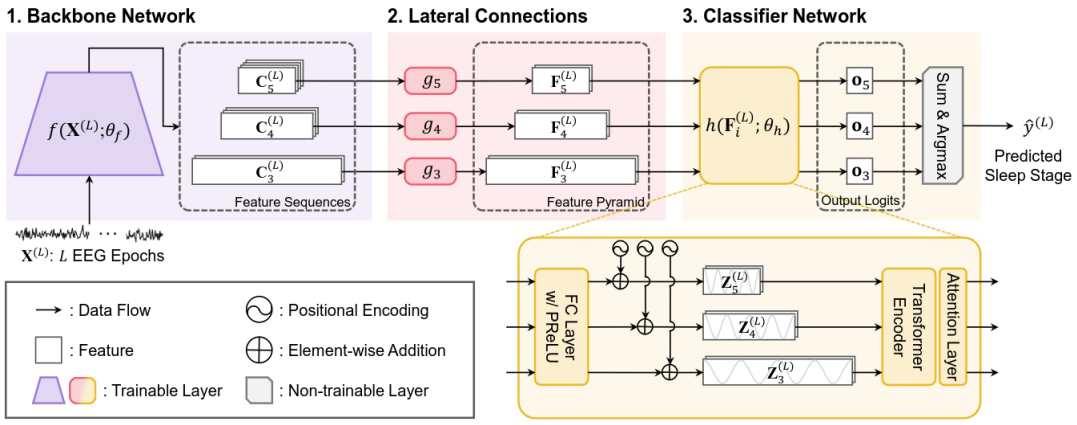


FIGURE 5.5 – Architecture de SleepyCo. Extrait de [93]

seule carte de caractéristiques. Ce processus crée une pyramide de caractéristiques. Elles sont obtenues par :

$$F_i^{(L)} = g_i(C_i^{(L)}; \theta_{g,i}) \tag{5.2}$$

avec $F_i^{(L)}$ la sortie appelée pyramide de caractéristiques, g_i la i-ème *lateral connection* et $\theta_{g,i}$ les paramètres. Dans SleepyCo, g est une couche de convolution. De manière similaire à AttentionSleep, les caractéristiques obtenues sont concaténées avec un bloc résiduel composé de couches denses. Enfin, un *Transformer* est utilisé pour représenter les relations temporelles entre les caractéristiques avant de prédire les états du sommeil via une couche dense. Nous utilisons l'implémentation [11] de SleepyCo fournie par [93] pour définir et apprendre le modèle. L'apprentissage est réalisé pendant 50 epochs avec un *early stopping* si le score de validation ne s'améliore pas pendant 20 epochs.

Bien que tous les modèles précédents basés sur des réseaux de neurones soient conçus pour des signaux 1D, nous changeons simplement la dimension d'entrée de la première couche de convolution pour utiliser nos 8 canaux conjointement.

5.5.2 Résultats

Nous calculons le score macro F1 (MF1) et l'*accuracy* obtenus par les différents modèles sur les 5 *folds* décrits dans le tableau 5.2, et présentons la moyenne obtenue sur ces 5 *folds*. Nous rappelons que des approches multidimensionnelles (comme [174]), et unidimensionnelles (comme [165], [54], [55]) obtiennent des résultats similaires sur les jeux de données SleepEDF et SHHS (voir section 2.2.3). Ainsi, nous présentons les scores de performance pour les modèles d'une part sur chaque canal individuel (c'est-à-dire avec l'entrée $x \in \mathbb{R}^{B\times3000\times1}$, B étant la taille du *batch*) et d'autre part avec les 8 canaux rassemblés ($x \in \mathbb{R}^{B\times3000\times8}$).

Les résultats sont présentés dans le tableau 5.3. Les meilleurs résultats sont obtenus par le modèle Sleep-Linear [174], basé sur le classifieur SGD en utilisant les 8 canaux simultanément, atteignant un score MF1 moyen de 0.658 et une *accuracy* moyenne de 0.738. Ces résultats sont en accord avec les excellentes performances de l'approche sur

11. https://github.com/gist-ailab/SleePyCo

les jeux SleepEDF et MASS en utilisant plusieurs canaux (voir le tableau 2.3).

Dans le cas unidimensionnel, SleepyCo [93] est la meilleure approche, obtenant les meilleurs scores sur les canaux O2-M1 (0.590/0.728), F4-M1 (0.612/0.748), F3-M2 (0.619/0.761) et E1-M2 (0.622/0.761). La performance de SleepyCo est moindre en utilisant les 8 canaux (0.585/0.725), ce qui est attendu car le modèle a été proposé pour fonctionner sur des signaux unidimensionnels. Enfin, utiliser l'information de tous les canaux donne la meilleure performance médiane (0.617/0.729), mais utiliser le canal F3-M2 offre des résultats compétitifs (0.608/0.732).

Méthode / Canal	C3-M2	C4-M1	O1-M2	O2-M1	F4-M1	F3-M2	E1-M2	E2-M1	Tous
Sleep-Linear(SGD)[174]	**0.619/0.716**	0.605/0.706	**0.591/0.686**	0.586/0.688	0.601/0.698	0.610/0.706	0.59/0.684	0.592/0.687	**0.658/0.738**
Sleep-Linear(CatBoost)[174]	0.616/0.689	**0.609/0.685**	0.587/0.656	0.587/0.660	0.606/0.673	0.607/0.676	0.597/0.664	0.590/0.669	0.647/0.713
AttentionSleep [54]	0.581/0.725	0.594/0.718	0.564/0.683	0.567/0.696	0.596/0.720	0.608/0.738	0.605/0.729	**0.600/0.721**	0.624/0.732
DeepSleepNet [165]	0.571/0.728	0.572/0.706	0.552/0.684	0.544/0.676	0.561/0.696	0.597/0.735	0.592/0.727	0.573/0.706	0.610/0.734
1D-CNN [55]	0.570/0.711	0.576/0.704	0.555/0.690	0.557/0.690	0.567/0.706	0.592/0.728	0.588/0.713	0.579/0.706	0.603/0.726
SleepyCo [93]	0.597/0.736	0.606/0.733	0.584/0.721	**0.590/0.728**	**0.612/0.748**	**0.619/0.761**	**0.622/0.761**	0.593/0.727	0.585/0.725

TABLE 5.3 – Macro F1/Accuracy de divers modèles en classification du sommeil sur le jeu COVISLEEP.

Ainsi, l'approche Sleep-Linear utilisant les 8 canaux conjointement obtient les meilleures performances en classification du sommeil. Puisque cette méthode est de plus interprétable, dans le sens de l'importance de caractéristiques, nous l'utiliserons pour apprendre à classifier les fenêtres de sommeil comme provenant d'un individu souffrant de COVID long ou non.

5.6 Identification du COVID long

Dans cette section, nous cherchons à identifier les différences entre les signaux de polysomnographie des patients souffrant d'insomnie chronique après contamination par le COVID long et des patients souffrant d'insomnie chronique non liée au COVID long. L'objectif n'est pas seulement la classification, mais également de pouvoir expliquer les différences entre les deux populations.

5.6.1 Statistiques de sommeil

Nous commençons par calculer diverses statistiques de macro-architecture du sommeil pour examiner si elles peuvent aider à distinguer entre les deux populations. Nous incluons des caractéristiques courantes du sommeil suivant [156] et [140] : Wake After Sleep Onset, Sleep Onset Latency, Sleep Efficiency, et les proportions en pourcentage de sommeil dans les états N1, N2, N3 et REM (on exclue ici les états d'éveil, puisque l'objectif est de déterminer un changement dans le sommeil, ainsi, les pourcentages sont recalculés de sorte que leur somme fasse 100 %.).

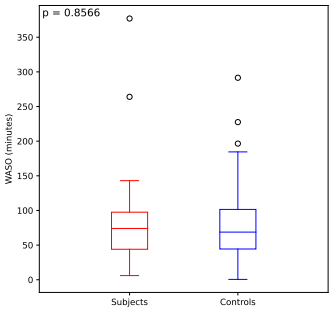
Nous calculons chaque statistique pour les patients infectés par le COVID-19 (sujets) et les patients non infectés (contrôles) et représentons ces statistiques sous forme de boîte à moustaches. Nous calculons également la valeur p correspondant au test de Student avec l'hypothèse que les deux populations sont identiques. Une valeur $p < 0.05$ rejette l'hypothèse et marque une distinction entre les deux populations.

Nous présentons les résultats dans la figure 5.6. Nous obtenons une valeur $p > 0.05$ pour chaque statistique testée, montrant la difficulté de distinguer les deux populations. Néanmoins, les sujets peuvent passer moins de temps en N3 lorsqu'ils dorment ($p = 0.204$) et plus en REM ($p = 0.226$). Nous examinerons les différences entre les stades du sommeil dans les sections suivantes.

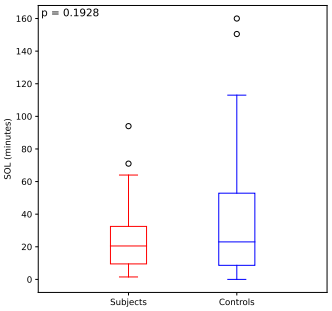
5.6.2 Classification

Ainsi, les statistiques de sommeil étudiées ne sont pas suffisantes à distinguer les populations. Nous abordons donc le problème sous l'angle de la classification binaire, où l'objectif est de trouver un modèle ζ tel que la prédiction $\hat{y} = \zeta(x)$ soit identique à la classe réelle y, avec ici $y \in \{0; 1\}$.

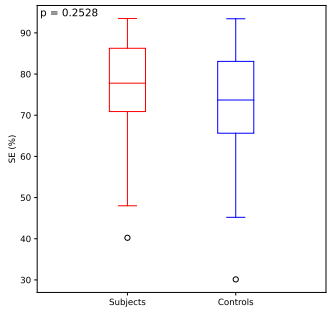
Nous prétraitons les données de la même manière que pour la classification du sommeil, mais définissons le label à 0 si la fenêtre provient d'un patient souffrant de COVID long, et à 1 sinon. Cependant, nous utilisons cette fois une validation "Leave One Out". Cela implique que nous apprenons le modèle sur tous les patients sauf 1, que nous utilisons



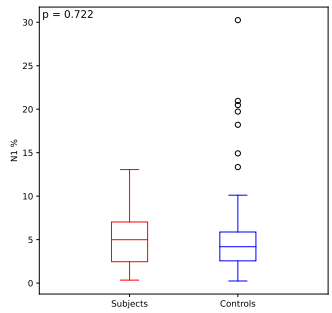
(a) Wake After Sleep Onset



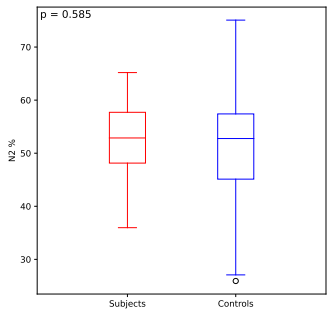
(b) Sleep Onset Latency



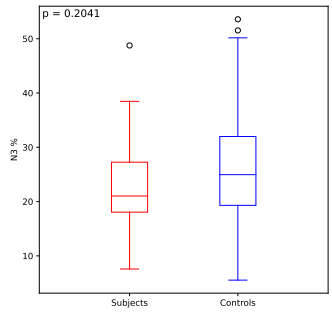
(c) Sleep Efficiency



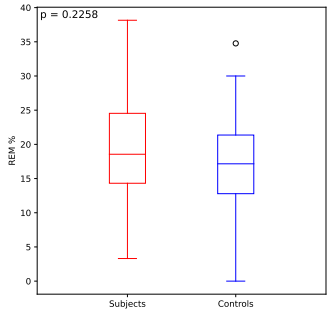
(d) Pourcentage de temps en N1



(e) Pourcentage de temps en N2



(f) Pourcentage de temps en N3



(g) Pourcentage de temps en REM

FIGURE 5.6 – Statistiques de sommeil des patients sujets et contrôles

pour évaluer le modèle. Nous répétons l'opération jusqu'à que tous les patients aient été utilisés comme jeu de test. Cela permet d'étudier la forte variabilité inter-patients de la détection de COVID. En effet, les modèles sont toujours appris sur tous les patients sauf 1. Ainsi, le jeu d'entraînement est très peu modifié entre les apprentissages, et donc les modèles appris sont très similaires, et la variation des performances sur les patients testés dépendra très probablement des signaux du patient concerné. Cela permet aussi de conserver un maximum de données pour l'entraînement. Nous utilisons ensuite le modèle Sleep-Linear mentionné précédemment pour apprendre la classification, en utilisant l'information des 8 canaux comme nous l'avons fait pour la classification du sommeil. Encore une fois, les caractéristiques sont générées en suivant [174].

Nous présentons les résultats dans le tableau 5.4 où nous faisons figurer la moyenne et l'écart type sur tous les patients et sur chacun des groupes, sujets et contrôles. Globalement, les résultats sont plutôt médiocres. Le meilleur score est obtenu par le modèle Sleep-Linear avec SGD comme classifieur, avec une accuracy moyenne de 0.572. L'écart type est très haut (0.257) ce qui illustre la grande variabilité entre les patients et la difficulté du modèle à généraliser. Sleep-Linear avec le CatBoost obtient la meilleure performance sur les sujets mais reste mauvais (accuracy moyenne de 0.421).

Nous illustrons cette grande variabilité inter-patients dans la figure 5.7. On rappelle que seulement 1/3 des personnes souffrent de COVID long, ce qui peut rendre la détection du COVID plus difficile. On affiche l'accuracy pour chaque patient, c'est-à-dire le ratio de fenêtres de signal bien classifiées. De manière générale, les sujets sont mal classifiés : seulement 11/31 (35.5 %) des sujets ont 50% de leurs fenêtres bien classifiées. Chez certains, l'impact du COVID long semble indétectable : 6/31 sujets (19.4%) ont une accuracy inférieure à 20%. Si la majorité (79%) des contrôles sont bien classifiés, 3 patients ont une accuracy inférieure à 20 %, suggérant peut-être une infection non déclarée du COVID.

La détection de marque de COVID à partir de fenêtre aléatoire apparait difficile et très variable entre les patients. Néanmoins, et comme nous l'avons mentionné dans la section 2.2, les différents états de sommeil sont caractérisés différemment. Nous recherchons donc dans la suite un potentiel impact plus marqué du COVID long dans certains états du sommeil.

Modèle	Tous	Sujets	Contrôles
Sleep-Linear+RF	**0.572 ± 0.257**	0.390 ± 0.234	**0.663 ± 0.216**
Sleep-Linear+SGD	0.493 ± 0.258	0.404 ± 0.245	0.538 ± 0.252
Sleep-Linear+CatBoost	0.556 ± 0.246	**0.421 ± 0.234**	0.623 ± 0.223

TABLE 5.4 – Accuracy en classification de signaux de sommeil, entre les signaux issus de personnes souffrant de COVID long (sujets) et les autres (contrôles).

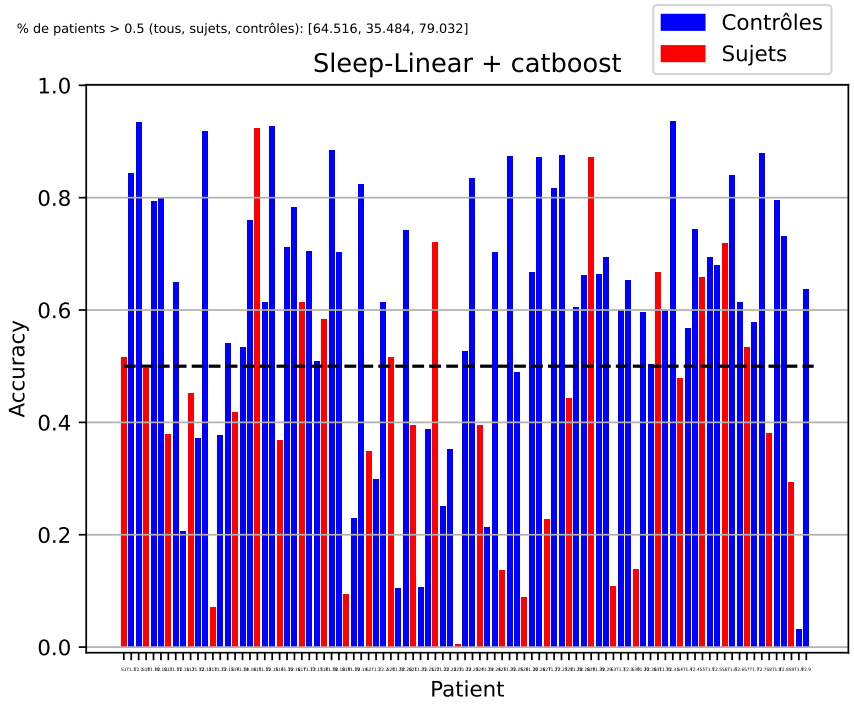


FIGURE 5.7 – Accuracy de la classification de signaux de sommeil par la méthode Sleep-Linear avec CatBoost, entre les signaux issus de personnes souffrant de COVID long (sujets) et les autres (contrôles), pour chaque patient. La ligne pointillée représente le seuil de 50% de précision (c'est-à-dire que 50% des fenêtres d'un individu sont bien classifiées).

Stage	Sleep-Linear+RF	Sleep-Linear+SGD	Sleep-Linear+CatBoost
Éveil	0.51 ± 0.277 | 0.374 ± 0.289 | 0.578 ± 0.244	0.276 ± 0.245 | 0.438 ± 0.304 | 0.196 ± 0.155	0.535 ± 0.254 | 0.317 ± 0.226 | 0.644 ± 0.189
N1	0.562 ± 0.334 | 0.435 ± 0.308 | 0.625 ± 0.328	0.446 ± 0.31 | 0.431 ± 0.259 | 0.454 ± 0.332	0.566 ± 0.318 | 0.391 ± 0.284 | 0.654 ± 0.298
N2	0.535 ± 0.32 | 0.375 ± 0.277 | 0.615 ± 0.310	0.474 ± 0.274 | 0.387 ± 0.276 | 0.517 ± 0.262	0.528 ± 0.308 | 0.376 ± 0.252 | 0.603 ± 0.305
N3	0.631 ± 0.307 | 0.437 ± 0.318 | 0.728 ± 0.251	0.489 ± 0.327 | 0.449 ± 0.332 | 0.509 ± 0.322	0.577 ± 0.350 | 0.413 ± 0.349 | 0.660 ± 0.319
REM	**0.637 ± 0.335** | 0.387 ± 0.341 | 0.764 ± 0.249	0.515 ± 0.330 | 0.451 ± 0.331 | 0.548 ± 0.325	0.59 ± 0.379 | 0.398 ± 0.388 | 0.688 ± 0.334

TABLE 5.5 – Accuracy de la classification de signaux de sommeil, entre les signaux issus de personnes souffrant de COVID long et les autres. Les signaux sont sélectionnés selon l'état de sommeil associé. On présente les résultats pour tous les patients, puis uniquement les sujets, puis uniquement les contrôles.

5.6.2.1 L'impact du COVID long est-il plus prévalent dans un stade de sommeil spécifique ?

Nous cherchons à déterminer si les caractéristiques distinctives du COVID long sont plus faciles à détecter dans certains stades du sommeil que dans d'autres. Pour ce faire, nous limitons l'ensemble de données aux fenêtres appartenant à un stade de sommeil spécifique, selon les labels affectés par les experts. Autrement dit, on imagine posséder des modèles reproduisant exactement la classification du sommeil manuelle, bien que ce ne soit pas le cas au vu des résultats du tableau 5.3. Nous reproduisons ensuite les expériences mentionnées dans le paragraphe précédent : nous apprenons à nouveau le classificateur à partir de zéro.

Nous présentons les résultats dans le tableau 5.5. L'impact du COVID long semble plus prévalent dans les stades N3 et REM, avec une *accuracy* plus élevée au stade REM pour à la fois le classificateur Random Forest (0.637) et CatBoost (0.590), bien que ce stade ne contienne qu'environ 13.4% des fenêtres. Le stade N3 semble aussi permettre de séparer certaines fenêtres, avec une *accuracy* moyenne de 0.631 via le Random Forest et de 0.577 via CatBoost. Ces résultats confirment l'intuition évoquée dans la figure 5.6 montrant que le sommeil en phase N3 et REM peut différer en cas d'insomnie induite par le COVID long. Ces résultats restent médiocres, avec notamment une accuracy sur les sujets inférieure à 0.5, et un écart type supérieur à 0.3.

Bien que nous ayons observé un impact plus marqué du COVID long sur les stades N3 et REM, cela reste limité par la forte variabilité des résultats. Nous nous penchons donc sur les causes de cette incohérence, en explorant si elle est due à la complexité de la tâche d'apprentissage ou aux caractéristiques individuelles des patients.

5.6.2.2 Variabilité inter-patients

Les scores des patients semblent varier significativement, sans que nous parvenions à expliquer pourquoi. Nous souhaitons déterminer si la variabilité est principalement due à la difficulté d'apprentissage de la tâche, ou si des patients présentent simplement des caractéristiques différentes de leur classe. Pour ce faire, nous nous intéressons d'abord à l'accord entre les prédictions selon les différents états de sommeil. Nous comparons les accuracy par patient obtenues via Sleep-Linear + RF sur les états N3 et REM, puisqu'on a obtenu les meilleurs scores sur ces états. Nous les nommons respectivement acc_{N3} et acc_{REM}, ce sont des vecteurs de $\mathbb{R}^{93}$.

Nous calculons le coefficient de corrélation de Pearson *pcc* entre les valeurs d'accuracy acc_{N3} et acc_{REM}. On rappelle que $pcc \in [-1, 1]$ et que -1 indique une corrélation complétement négative (les vecteurs comparés évoluent inversement), 1 une corrélation parfaite et 0 une absence de corrélation. On obtient $pcc(acc_{N3}, acc_{REM}) = 0.23$: les prédictions réalisées aux états N3 et REM ne semblent pas globalement corrélées. Pour entrer dans le détail, nous choisissons un seuil T. Nous définissons l'accord entre les prédictions de la manière suivante : si ($acc_{N3} > T$ et $acc_{REM} > T$) ou ($acc_{N3} < 1 - T$ et $acc_{REM} < 1 - T$) alors les prédictions sont en accord.

En choisissant $T = 0.5$, les prédictions acc_{N3} et acc_{REM} sont en accord sur 60 des 93

patients (64.5%), dont 12/31 (38.7%) des sujets et donc 48/62 (77.4 %) des contrôles. Ainsi, les prédictions des états N3 et REM, en particulier concernant les sujets, ne semblent pas similaires. Nous étudions dans la suite si la variabilité peut s'expliquer par la durée depuis l'infection.

Selon les symptômes, l'impact du COVID long diminue ou s'aggrave avec le temps [48]. Puisque les dates de contamination ont été recueillies dans le jeu de données, nous étudions la corrélation entre la durée depuis l'infection et l'accuracy obtenue par le modèle sur les patients concernés.

Nous calculons la durée entre la contamination et l'examen pour chaque patient et obtenons en moyenne une durée de 441.96 ± 234.37 jours. En utilisant les prédictions réalisées par Sleep-Linear + RF sur l'état N3 acc_{N3}, on obtient $pcc = -0.01$. Ainsi, les prédictions ne semblent pas corrélées à la durée depuis l'infection. Si l'on restreint acc_{N3} aux 12 patients pour lesquels acc_{N3} et acc_{REM} étaient d'accord précédemment, on obtient $pcc = -0.23$, ce qui reste largement insuffisant pour supposer une quelconque corrélation. La facilité à détecter l'impact du COVID long ne semble ainsi pas liée à la durée depuis l'infection.

5.7 Conclusion du chapitre

Le COVID long a significativement dégradé le sommeil de nombreuses personnes, la majorité souffrant d'insomnie chronique. L'élaboration d'un traitement adapté nécessite de comprendre les causes de cette insomnie, et notamment si elle est due au stress causé par l'infection, ou si elle résulte de modifications des signaux cérébraux après l'infection.

Pour ce faire, nous avons méticuleusement collecté les polysomnographies de personnes souffrant d'insomnie chronique, en séparant celles déclarant souffrir de COVID long des autres. Nous avons ainsi formé un jeu de données contenant les enregistrements de 93 patients, dont 31 souffrant d'insomnie chronique après une infection du SARS-CoV-2. Nous mettons le jeu de données à disposition de la communauté pour aider à l'avancement de la compréhension des mécanismes engendrés par le COVID long sur le sommeil.

De plus, nous avons appliqué divers algorithmes de classification du sommeil à l'état de l'art et mis en avant la relative difficulté de la tâche sur le jeu COVISLEEP. Nous avons montré que l'approche Sleep-Linear [174], basée sur une création de caractéristiques manuelle, obtient les meilleures performances. Ces résultats restent largement inférieurs à ceux obtenus sur d'autres jeux de données usuels du sommeil (SleepEDF [83], SHHS [186], DOD-O [67], MASS [116]), illustrant la significative marge d'amélioration. Nous espérons que la mise à disposition de COVISLEEP permettra à la communauté d'étudier les raisons de cet écart de performance.

Enfin, nous avons étudié la détection de l'impact du COVID. Nous avons montré que des statistiques de macro-architecture de sommeil ne suffisent pas à distinguer les polysomnographies des sujets de celles des contrôles, confirmant les résultats de

[140]. Nous avons ensuite utilisé Sleep-Linear pour apprendre la détection de COVID long par classification, ce qui a mis en avant la difficulté de la tâche, notamment à cause d'une forte variabilité entre les patients. Enfin, nous avons illustré que l'impact du COVID long semble être plus marqué sur les états de sommeil N3 et REM. Cela reste cependant largement dépendant des patients, du fait de la forte variabilité. Nous émettons l'hypothèse que les caractéristiques choisies par la méthode Sleep-Linear ne suffisent pas à détecter l'information de COVID pour tous les patients, et encourageons la communauté à rechercher d'autres caractéristiques, par exemple en modifiant les fonctions d'extraction de caractéristiques, ou en utilisant des fenêtres plus longues, ou d'autres canaux.

Chapitre 6

Conclusion et Perspectives

Dans cette thèse, nous nous sommes intéressés à la classification et à la comparaison de séries temporelles, en spécifiant notre étude aux signaux de sommeil. Nous avons montré que ces signaux, nécessaires à l'étude de pathologies comme l'insomnie, sont volumineux. Pour diminuer les coûts de stockage et de transfert de ces données, nous nous sommes intéressés à la compression de ces signaux de sommeil. Notamment grâce aux autoencodeurs, différentes approches permettent d'atteindre de très hauts taux de compression sur les signaux de santé. Néanmoins, ces approches utilisent principalement des métriques basées sur la distance euclidienne pour valider la qualité de la reconstruction, et comparent donc les signaux reconstruits avec les signaux originaux au moyen de cette distance. Or, de nombreux travaux que nous avons présentés dans ce manuscrit illustrent que des métriques adaptées aux séries temporelles, en particulier DTW, permettent des comparaisons de séries plus adaptées aux tâches de requêtage ou de classification. DTW n'étant pas différentiable, il est impossible de l'utiliser comme fonction de coût pour l'apprentissage d'un autoencodeur.

Pour résoudre ce problème et en se basant sur les contributions de [40] et [100], nous avons proposé deux méthodes d'approximation de DTW nommées DeepDTW. La première utilise un réseau siamois pour encoder les signaux de sorte que la distance euclidienne des signaux projetés soit le plus proche possible de la DTW entre les signaux originaux. Elle permet notamment de comparer des signaux de tailles différentes, comme DTW. La deuxième approche concatène simplement les signaux d'entrée pour tenter de prédire la valeur de DTW. Nous avons montré au travers de tâches de recherche de séries similaires et de classification d'état du sommeil que nos méthodes obtiennent des résultats similaires aux autres approximations de DTW, tout en étant différentiable et de complexité linéaire par rapport à la longueur des séries, élément important pour la compression de longues séries temporelles. Enfin, nous avons montré que nos approches peuvent généraliser à d'autres jeux de données du sommeil sans perte de performance.

Nous avons ensuite utilisé ces approximations pour l'apprentissage d'un autoencodeur permettant de compresser des signaux de sommeil. En utilisant le TCN-AE [170] nous pouvons ainsi faire varier le taux de compression. Nous avons illustré que l'utilisa-

tion de DeepDTW comme fonction de coût de reconstruction permet d'obtenir de meilleures performances de classification que la MSE ou softDTW sur les données reconstruites. Nous avons aussi montré que notre compression permet de mieux classifier les données reconstruites que des méthodes de compression basées sur DCT, DWT ou FFT. Enfin, et de manière similaire à nos travaux sur l'approximation de DTW, nous avons montré que notre système est capable de compresser des données d'un autre jeu d'enregistrement de sommeil sans pertes de performance et sans *finetuning*.

Toujours dans le cadre de l'analyse de données de sommeil, nous avons ensuite étudié l'impact du COVID long sur l'insomnie au cours d'une collaboration entre Saagie, VIFASOM et l'IRBA : le projet PANDORE-IA. En effet, bien que diverses études aient illustré l'augmentation du risque d'insomnie chez les personnes souffrant de COVID long, celle-ci ne semble pas distinguable de l'insomnie chronique usuelle. Nous avons ainsi, en partenariat avec VIFASOM, construit et mis à disposition de la communauté un jeu de données appelé COVISLEEP comprenant les polysomnographies de personnes ayant développé une insomnie après infection du COVID, ainsi que les polysomnographies de personnes insomniaques n'ayant pas été infectées par le virus. Nous avons ensuite confirmé que les deux populations ne sont pas distinguables via les caractéristiques habituellement utilisées pour caractériser les signaux de sommeil, puis avons comparé divers modèles sur la classification de sommeil de COVISLEEP. Nous avons ainsi identifié que l'approche Sleep-Linear [174] obtient les meilleures performances. De manière surprenante, ces performances sont significativement inférieures à celles obtenues sur les jeux de données de sommeil usuels (SleepEDF [83], SHHS [186], DOD-O [67], MASS [116], ...). En mettant le jeu de données COVISLEEP à disposition de la communauté, nous espérons que cela motivera l'étude des raisons de cet écart.

Enfin, nous nous sommes penchés sur l'apprentissage d'un modèle capable de détecter le COVID long en utilisant l'approche Sleep-Linear. Nous avons illustré la complexité de la tâche, notamment due à la forte variabilité entre les patients. Nous avons montré que cette variance ne semble ni due à l'état du sommeil des signaux considérés, ni à la durée entre la contamination et l'enregistrement. Nous encourageons la communauté à s'intéresser aux raisons expliquant cette variance. Nous émettons l'hypothèse que les mauvaises performances en détection du COVID long sont dues au fait que les caractéristiques choisies par Sleep-Linear ne permettent pas d'extraire l'information de COVID pour tous les patients. Les fenêtres temporelles choisies pourraient aussi être trop courtes, ou l'impact du COVID pourrait être plus marqué dans d'autres canaux que ceux choisis durant notre étude.

Nos contributions à l'approximation de DTW, la compression de signaux de sommeil et la détection de COVID comportent cependant certaines limites.

6.1 Perspectives

6.1.1 Approches multidimensionnelles et généralisation

Bien que notre méthode siamoise d'approximation de DTW permette de comparer des signaux de longueurs différentes, elle est limitée aux signaux unidimensionnels. Cela a un faible impact dans le cas des signaux de sommeil, car les scores de classification avec des signaux 1D sont généralement similaires à ceux obtenus via des approches multidimensionnelles. Cela limite néanmoins les capacités de généralisation de nos modèles. Ainsi, il serait intéressant d'adapter nos approximations pour pouvoir comparer des séries multidimensionnelles. En particulier, cela consisterait à remplacer le SorsNet utilisé pour encoder les signaux par un réseau capable d'encoder des séries indépendamment de leur nombre de dimensions, tout en conservant la capacité du SorsNet à encoder des séries de différentes longueurs. Par exemple, le RobustSleepNet [66] permet de classifier l'état du sommeil de séries en créant l'architecture sans devoir fixer le nombre de dimensions. Il utilise malheureusement des couches récurrentes, ce qui n'est pas compatible avec notre volonté de limiter la complexité temporelle. Ainsi, il serait intéressant de rechercher une architecture satisfaisant tous les critères précédents, tout en conservant les excellentes capacités de classification des signaux du sommeil démontrées par RobustSleepNet. Cela permettrait de plus d'étudier les capacités de généralisation de nos méthodes sur le jeu de données COVISLEEP.

Obtenir un modèle capable d'approximer DTW indépendamment de la longueur et du nombre de dimensions des signaux serait un pas vers une approximation généraliste de DTW, c'est-à-dire capable d'approximer DTW pour tous types de séries temporelles. En effet, nous pourrions alors mélanger différents types de séries temporelles dans le jeu de données d'apprentissage et évaluer nos approximations sur la classification de ces différents types de séries. En utilisant le même raisonnement, cela permettrait d'avancer vers un système de compression des séries temporelles basé sur un réseau de neurones indépendant du type de séries utilisées.

En plus des pistes évoquées précédemment, une manière d'améliorer les performances sur les tâches de classification et de détection de COVID est d'essayer de comprendre et d'expliquer les décisions et les erreurs du modèle.

6.1.2 Explicabilité de la détection du COVID long

L'explicabilité des modèles implique la compréhension de la manière dont un modèle parvient à classifier un signal dans un état de sommeil donné. C'est important dans le contexte de classification de données de santé pour les raisons suivantes :

— Adoption par les médecins : un système de santé n'est utile que s'il est utilisé. Pour faciliter son adoption par le corps médical, avoir un système compréhensible et transparent accroit la confiance des médecins et les chances d'utilisation du système. [136]

— Vérification : en comprenant les décisions faites par le modèle, les experts peuvent vérifier la validité de celles-ci avec leurs propres connaissances, augmentant ainsi indirectement les performances du modèle. [162]

— Améliorations du modèle : comprendre les décisions faites par le modèle permet de comprendre ses erreurs et donc d'aider à les résoudre. [2]

Malgré l'importance de la nécessité de pouvoir expliquer les décisions prises par les modèles, peu de travaux explorent le problème dans le contexte de la classification du sommeil. Des approches comme [3], qui combinent les performances des réseaux de neurones et l'explicabilité des règles expertes dans la classification du sommeil en utilisant l'apprentissage de prototypes, pourraient permettre une réelle utilisation des algorithmes de classification de sommeil par des experts de VIFASOM. Cela pourrait aussi permettre de mieux comprendre la forte variabilité entre les patients que nous avons identifiée lors de l'apprentissage de la détection du COVID long. En particulier, cela permettrait de comprendre les similarités entre les patients bien classifiés et ainsi de justifier la difficulté à bien classifier les autres. Les travaux de [3] restent cependant limités par les performances inférieures à l'état de l'art en classification du sommeil de leurs méthodes.

Nous sommes intéressés de suivre l'évolution de l'analyse des signaux de santé dans les années futures, en particulier sous les angles de la capacité à généraliser et de l'explicabilité des modèles.

Liste des communications

Articles

Lerogeron Hugo, et al. "Approximating dynamic time warping with a convolutional neural network on EEG data." Pattern Recognition Letters 171 (2023) : 162-169.

Lerogeron Hugo, et al. "Learning an autoencoder to compress EEG signals via a neural network based approximation of DTW." Procedia Computer Science 222 (2023) : 448-457.

Lerogeron Hugo, et al. "COVISLEEP : A sleep dataset to investigate the impact of long-COVID on chronic insomnia.", *under review.*

Posters

Benballa Miriam, et al. "Prédiction de la performance de conduite sur simulateur à l'aide de paramètres cliniques et de sommeil : le Projet PANDORE-IA", World Sleep 2023

Lerogeron Hugo, et al. "An AI-Based Approach for Detecting long COVID-19 Patients through Sleep Polysomnography Analysis : The Pandore-IA project", World Sleep 2023

Communications orales

Lerogeron Hugo, et al. "Learning an autoencoder to compress EEG signals via a neural network based approximation of DTW.", *vidéo* , International Neural Network Society Workshop on Deep Learning Innovations and Applications 2023

Lerogeron Hugo, et al. "An AI-Based Approach for Detecting long COVID-19 Patients through Sleep Polysomnography Analysis : The Pandore-IA project", Le Congrès du Sommeil 2023

Bibliographie

[1] Amaia Abanda, Usue Mori, and Jose A Lozano. A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2) :378–412, 2019.

[2] Amina Adadi and Mohammed Berrada. Explainable ai for healthcare : from black box to interpretable models. In *Embedded Systems and Artificial Intelligence : Proceedings of ESAI 2019, Fez, Morocco*, pages 327–337. Springer, 2020.

[3] Irfan Al-Hussaini and Cassie S Mitchell. Performance and utility trade-off in interpretable sleep staging. *arXiv preprint arXiv:2211.03282*, 2022.

[4] Abeer Z Al-Marridi, Amr Mohamed, and Aiman Erbad. Convolutional autoencoder approach for eeg compression and reconstruction in m-health systems. In *2018 14th international wireless communications & mobile computing conference (IWCMC)*, pages 370–375. IEEE, 2018.

[5] Ali Ameri, Mohammad Ali Akhaee, Erik Scheme, and Kevin Englehart. Regression convolutional neural network for improved simultaneous emg control. *Journal of neural engineering*, 16(3) :036015, 2019.

[6] Hussein El Amouri, Thomas Lampert, Pierre Gançarski, and Clément Mallet. Cdps : Constrained dtw-preserving shapelets. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 21–37. Springer, 2022.

[7] Allan Rechtschaffen Anthony Kales. *A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects*. United States Government Printing Office, 1968.

[8] M Arunasalam, N Yaakob, A Amir, M Elshaikh, and NF Azahar. Real-time drowsiness detection system for driver monitoring. In *IOP Conference Series : Materials Science and Engineering*, volume 767, page 012066. IOP Publishing, 2020.

[9] Anthony Bagnall and Gareth Janacek. A run length transformation for discriminating between auto regressive time series. *Journal of classification*, 31 :154–178, 2014.

[10] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off : a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, 31 :606–660, 2017.

[11] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[13] Aranka V Ballering, Sander KR van Zon, Tim C Olde Hartman, and Judith GM Rosmalen. Persistence of somatic symptoms after covid-19 in the netherlands : an observational cohort study. *The Lancet*, 400(10350) :452–461, 2022.

[14] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[15] Parikshit Bansal, Prathamesh Deshpande, and Sunita Sarawagi. Missing value imputation on multidimensional time series. *arXiv preprint arXiv:2103.01600*, 2021.

[16] Vishal Barot and Ritesh Patel. A physiological signal compression approach using optimized spindle convolutional auto-encoder in mhealth applications. *Biomedical Signal Processing and Control*, 73 :103436, 2022.

[17] Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving the kolmogorov pde by means of deep learning. *Journal of Scientific Computing*, 88(3), 2021.

[18] Richard Bellman. Dynamic programming. *Science*, 153(3731) :34–37, 1966.

[19] Miriam Benballa. *Analyse de sentiments sur Twitter dans un contexte faiblement supervisé*. PhD thesis, Normandie Université, 2022.

[20] Julius Berner, Markus Dablander, and Philipp Grohs. Numerically solving parametric families of high-dimensional kolmogorov partial differential equations via deep learning. *Advances in Neural Information Processing Systems*, 33 : 16615–16627, 2020.

[21] Jan Blechschmidt and Oliver G Ernst. Three ways to solve partial differential equations with neural networks—a review. *GAMM-Mitteilungen*, 44(2) : e202100006, 2021.

[22] Mathieu Blondel, Arthur Mensch, and Jean-Philippe Vert. Differentiable divergences between time series. In *International Conference on Artificial Intelligence*

and Statistics, pages 3853–3861. PMLR, 2021.

[23] Leo Bluestein. A linear filtering approach to the computation of discrete fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4) :451–455, 1970.

[24] J Frédéric Bonnans and Alexander Shapiro. Optimization problems with perturbations : A guided tour. *SIAM review*, 40(2) :228–264, 1998.

[25] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[26] Leo Breiman. Random forests. *Machine learning*, 45 :5–32, 2001.

[27] Clemens Brunner, Robert Leeb, Gernot Müller-Putz, Alois Schlögl, and Gert Pfurtscheller. Bci competition 2008–graz data set a. *Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology*, 16 :1–6, 2008.

[28] Frederick W Byron and Robert W Fuller. *Mathematics of classical and quantum physics*. Courier Corporation, 2012.

[29] Xingyu Cai, Tingyang Xu, Jinfeng Yi, Junzhou Huang, and Sanguthevar Rajasekaran. Dtwnet : a dynamic time warping network. *Advances in Neural Information Processing Systems*, 32, 2019.

[30] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits : Bidirectional recurrent imputation for time series. *Advances in neural information processing systems*, 31, 2018.

[31] Youshen Cao, Hanzhi Zhang, Yong-Bae Choi, Hao Wang, and Sicheng Xiao. Hybrid deep learning model assisted data compression and classification for efficient data delivery in mobile health applications. *IEEE Access*, 8 :94757–94766, 2020.

[32] Stanislas Chambon, Mathieu N Galtier, Pierrick J Arnal, Gilles Wainrib, and Alexandre Gramfort. A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 26(4) :758–769, 2018.

[33] Xiaobin Chang, Frederick Tung, and Greg Mori. Learning discriminative prototypes with dynamic time warping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8395–8404, 2021.

[34] Giacomo Chiarot and Claudio Silvestri. Time series compression survey. *ACM Computing Surveys*, 55(10) :1–32, 2023.

[35] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau,

Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[36] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[37] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1) :37–46, 1960.

[38] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90) :297–301, 1965.

[39] Denis Coquenet. *Vers l'apprentissage de modèles de bout-en-bout pour la reconnaissance de documents manuscrits*. PhD thesis, Rouen University, 2022. URL http://www.theses.fr/2022NORMR028. Thèse de doctorat dirigée par Paquet, Thierry Informatique Normandie 2022.

[40] Nicolas Courty, Rémi Flamary, and Mélanie Ducoffe. Learning wasserstein embeddings, 2017.

[41] Marco Cuturi and Mathieu Blondel. Soft-dtw : a differentiable loss function for time-series. In *International conference on machine learning*, pages 894–903. PMLR, 2017.

[42] Evangelin Dasan and Rajakumar Gnanaraj. Joint ecg–emg–eeg signal compression and reconstruction with incremental multimodal autoencoder approach. *Circuits, Systems, and Signal Processing*, 41(11) :6152–6181, 2022.

[43] Evangelin Dasan and Ithayarani Panneerselvam. A novel dimensionality reduction approach for ecg signal via convolutional denoising autoencoder with lstm. *Biomedical Signal Processing and Control*, 63 :102225, 2021.

[44] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6) :1293–1305, 2019.

[45] Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

[46] Michael E Davies and Thomas Blumensath. Faster & greedier : algorithms for sparse reconstruction of large datasets. In *2008 3rd International Symposium on Communications, Control and Signal Processing*, pages 774–779. IEEE, 2008.

[47] Hannah E Davis, Gina S Assaf, Lisa McCorkell, Hannah Wei, Ryan J Low, Yochai Re'em, Signe Redfield, Jared P Austin, and Athena Akrami. Characterizing

long covid in an international cohort : 7 months of symptoms and their impact. *EClinicalMedicine*, 38, 2021.

[48] Hannah E Davis, Lisa McCorkell, Julia Moore Vogel, and Eric J Topol. Long covid : major findings, mechanisms and recommendations. *Nature Reviews Microbiology*, 21(3) :133–146, 2023.

[49] Luigi De Gennaro and Michele Ferrara. Sleep spindles : an overview. *Sleep medicine reviews*, 7(5) :423–440, 2003.

[50] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi : 10.1109/CVPR.2009.5206848.

[51] Kimia Dinashi, Ali Ameri, Mohammad Ali Akhaee, Kevin Englehart, and Erik Scheme. Compression of emg signals using deep convolutional autoencoders. *IEEE Journal of Biomedical and Health Informatics*, 2022.

[52] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data : experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1 (2) :1542–1552, 2008.

[53] Oussama El B'charri, Rachid Latif, Wissam Jenkal, and Abdenbi Abenaou. The ecg signal compression using an efficient algorithm based on the dwt. *International Journal of Advanced Computer Science and Applications*, 7(3), 2016.

[54] Emadeldeen Eldele, Zhenghua Chen, Chengyu Liu, Min Wu, Chee-Keong Kwoh, Xiaoli Li, and Cuntai Guan. An attention-based deep learning approach for sleep stage classification with single-channel eeg. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29 :809–818, 2021.

[55] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-series representation learning via temporal and contextual contrasting. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pages 2352–2359*, 2021.

[56] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee-Keong Kwoh, and Xiaoli Li. Self-supervised learning for label-efficient sleep stage classification : A comprehensive evaluation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 31 :1333–1342, 2023.

[57] Johann Faouzi. Time series classification : A review of algorithms and implementations. *Machine Learning (Emerging Trends and Applications)*, 2022.

[58] Hassan Ismail Fawaz. *Deep Learning for Time Series Classification*. PhD thesis, Université de Haute-Alsace, 2020. URL https://theses.hal.science/tel-0371501

6/preview/2020MULH3604_these_ISMAIL_FAWAZ.pdf.

[59] Christian Garbin, Xingquan Zhu, and Oge Marques. Dropout vs. batch normalization : an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, 79 :12777–12815, 2020.

[60] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1) :1–58, 1992.

[61] Ramazan Gençay and Min Qi. Pricing and hedging derivative securities with neural networks : Bayesian regularization, early stopping, and bagging. *IEEE Transactions on Neural Networks*, 12(4) :726–734, 2001.

[62] AKSU Gökhan, Cem Oktay Güzeller, and Mehmet Taha Eser. The effect of the normalization method used in different sample sizes on the success of artificial neural network model. *International Journal of Assessment Tools in Education*, 6(2) :170–192, 2019.

[63] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly : Memory-augmented deep autoencoder for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1705–1714, 2019.

[64] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[65] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401, 2014.

[66] Antoine Guillot and Valentin Thorey. Robustsleepnet : Transfer learning for automated sleep staging at scale. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29 :1441–1451, 2021.

[67] Antoine Guillot, Fabien Sauvet, Emmanuel H During, and Valentin Thorey. Dreem open datasets : Multi-scored sleep datasets to compare human and automated sleep staging. *IEEE transactions on neural systems and rehabilitation engineering*, 28(9) :1955–1965, 2020.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers : Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer*

vision and pattern recognition, pages 770–778, 2016.

[70] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[71] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data mining and knowledge discovery*, 28 :851–881, 2014.

[72] Daniel S Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4) :664–675, 1977.

[73] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[74] Conrad Iber, Sonia Ancoli-Israel, Andrew L Chesson, Stuart F Quan, et al. *The AASM manual for the scoring of sleep and associated events : rules, terminology and technical specifications*, volume 1. American academy of sleep medicine Westchester, IL, 2007.

[75] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification : a review. *Data mining and knowledge discovery*, 33(4) :917–963, 2019.

[76] Vugar E Ismailov. On the approximation by neural networks with bounded number of neurons in hidden layers. *Journal of Mathematical Analysis and Applications*, 417(2) :963–969, 2014.

[77] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on acoustics, speech, and signal processing*, 23 (1) :67–72, 1975.

[78] Weiwei Jiang. Time series classification : Nearest neighbor versus deep learning models. *SN Applied Sciences*, 2(4) :721, 2020.

[79] Barry L Kalman and Stan C Kwasny. Why tanh : choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 578–581. IEEE, 1992.

[80] Samsul Ariffin Abdul Karim, Mohd Hafizi Kamarudin, Bakri Abdul Karim, Mohammad Khatim Hasan, and Jumat Sulaiman. Wavelet transform and fast fourier transform for signal compression : A comparative study. In *2011 International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, pages 280–285. IEEE, 2011.

[81] Harleen Kaur, Benjamin C Spurling, and Pradeep C Bollu. Chronic insomnia. 2018.

[82] Hüseyin Kaya and Şule Gündüz-Öğüdücü. A distance based time series classification framework. *Information Systems*, 51 :27–42, 2015.

[83] B. Kemp, A.H. Zwinderman, B. Tuk, H.A.C. Kamphuisen, and J.J.L. Oberye. Analysis of a sleep-dependent neuronal feedback loop : the slow-wave microcontinuity of the eeg. *IEEE Transactions on Biomedical Engineering*, 47(9) : 1185–1194, 2000. doi : 10.1109/10.867928.

[84] Eamonn Keogh, Li Wei, Xiaopeng Xi, Michail Vlachos, Sang-Hee Lee, and Pavlos Protopapas. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures. *The VLDB journal*, 18 :611–630, 2009.

[85] Sirvan Khalighi, Teresa Sousa, José Moutinho Santos, and Urbano Nunes. Isruc-sleep : A comprehensive public dataset for sleep researchers. *Computer methods and programs in biomedicine*, 124 :180–192, 2016.

[86] Abdelouahab Khelifati, Mourad Khayati, and Philippe Cudré-Mauroux. Corad : correlation-aware compression of massive time series using sparse dictionary coding. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2289–2298. IEEE, 2019.

[87] Sang-Wook Kim, Sanghyun Park, and Wesley W Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings 17th international conference on data engineering*, pages 607–614. IEEE, 2001.

[88] Diederik P Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[89] Sander Koelstra, Christian Muhl, Mohammad Soleymani, Jong-Seok Lee, Ashkan Yazdani, Touradj Ebrahimi, Thierry Pun, Anton Nijholt, and Ioannis Patras. Deap : A database for emotion analysis ; using physiological signals. *IEEE transactions on affective computing*, 3(1) :18–31, 2011.

[90] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1) :79–86, 1951.

[91] Tuong Ly Le and Minh-Huan Vo. Lossless data compression algorithm to save energy in wireless sensor network. In *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, pages 597–600. IEEE, 2018.

[92] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4) :541–551, 1989.

[93] Seongju Lee, Yeonguk Yu, Seunghyeok Back, Hogeon Seo, and Kyoobin Lee. Sleepyco : Automatic sleep scoring with feature pyramid and contrastive lear-

ning. *arXiv preprint arXiv:2209.09452*, 2022.

[94] Daoyuan Li, Tegawendé François D Assise Bissyande, Jacques Klein, and Yves Le Traon. Time series classification with discrete wavelet transformed data : Insights from an empirical study. In *The 28th international conference on software engineering and knowledge engineering (SEKE 2016)*, 2016.

[95] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[96] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29 :565–592, 2015.

[97] Michael Littner, Max Hirshkowitz, Milton Kramer, Sheldon Kapen, W McDowell Anderson, Dennis Bailey, Richard B Berry, David Davila, Stephen Johnson, Clete Kushida, et al. Practice parameters for using polysomnography to evaluate insomnia : an update. *Sleep*, 26(6) :754–760, 2003.

[98] Deland Hu Liu and Syed Anas Imtiaz. Studying the effects of compression in eeg-based wearable sleep monitoring systems. *IEEE Access*, 8 :168486–168501, 2020.

[99] Pengfei Liu, Xiaoming Sun, Yang Han, Zhishuai He, Weifeng Zhang, and Chenxu Wu. Arrhythmia classification of lstm autoencoder based on time series anomaly detection. *Biomedical Signal Processing and Control*, 71 :103228, 2022.

[100] Arnaud Lods, Simon Malinowski, Romain Tavenard, and Laurent Amsaleg. Learning dtw-preserving shapelets. In *Advances in Intelligent Data Analysis XVI : 16th International Symposium, IDA 2017, London, UK, October 26–28, 2017, Proceedings 16*, pages 198–209. Springer, 2017.

[101] Alexander Malafeev, Dmitry Laptev, Stefan Bauer, Ximena Omlin, Aleksandra Wierzbicka, Adam Wichniak, Wojciech Jernajczyk, Robert Riener, Joachim Buhmann, and Peter Achermann. Automatic human sleep stage scoring using deep neural networks. *Frontiers in neuroscience*, 12 :781, 2018.

[102] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[103] Alice Marascu, Pascal Pompey, Eric Bouillet, Michael Wurst, Olivier Verscheure, Martin Grund, and Philippe Cudre-Mauroux. Tristan : Real-time analytics on massive time series using sparse dictionary compression. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 291–300. IEEE, 2014.

[104] John Marshall, Alistair Adcroft, Chris Hill, Lev Perelman, and Curt Heisey. A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research : Oceans*, 102(C3) : 5753–5766, 1997.

[105] Pierre-François Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE transactions on pattern analysis and machine intelligence*, 31(2) :306–318, 2008.

[106] GC Mbata and JC Chukwuka. Obstructive sleep apnea hypopnea syndrome. *Annals of medical and health sciences research*, 2(1) :74–77, 2012.

[107] Martin McKinney and Jeroen Breebaart. Features for audio and music classification. 2003.

[108] Ilona Merikanto, Yves Dauvilliers, Frances Chung, Yun Kwok Wing, Luigi De Gennaro, Brigitte Holzinger, Bjørn Bjorvatn, Charles M Morin, Thomas Penzel, Christian Benedict, et al. Sleep symptoms are essential features of long-covid–comparing healthy controls with covid-19 cases of different severity in the international covid sleep study (icoss-ii). *Journal of Sleep Research*, 32(1) : e13754, 2023.

[109] Jan-Gerd Meß, Robert Schmidt, Goerschwin Fey, and Frank Dannemann. On the compression of spacecraft housekeeping data using discrete cosine transforms. In *2016 International Workshop on Tracking, Telemetry and Command Systems for Space Applications (TTC)*, pages 1–8. IEEE, 2016.

[110] Jianliang Min, Ming Cai, Chao Gou, Chen Xiong, and Xuejiao Yao. Fusion of forehead eeg with machine vision for real-time fatigue detection in an automatic processing pipeline. *Neural Computing and Applications*, 35(12) :8859–8872, 2023.

[111] George B Moody and Roger G Mark. The impact of the mit-bih arrhythmia database. *IEEE engineering in medicine and biology magazine*, 20(3) :45–50, 2001.

[112] Alissa Elen Formiga Moura, Danilo Nunes Oliveira, Danielle Mesquista Torres, José Wagner Leonel Tavares-Júnior, Paulo Ribeiro Nóbrega, Pedro Braga-Neto, and Manoel Alves Sobreira-Neto. Central hypersomnia and chronic insomnia : expanding the spectrum of sleep disorders in long covid syndrome-a prospective cohort study. *BMC neurology*, 22(1) :1–10, 2022.

[113] Sajad Mousavi, Fatemeh Afghah, and U Rajendra Acharya. Sleepeegnet : Automated sleep stage scoring with sequence to sequence deep learning approach. *PloS one*, 14(5) :e0216456, 2019.

[114] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on*

machine learning (ICML-10), pages 807–814, 2010.

[115] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions : Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[116] Christian O'reilly, Nadia Gosselin, Julie Carrier, and Tore Nielsen. Montreal archive of sleep studies : an open-access resource for instrument benchmarking and exploratory research. *Journal of sleep research*, 23(6) :628–635, 2014.

[117] Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. In *Computer Vision–ACCV 2016 : 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II 13*, pages 189–204. Springer, 2017.

[118] Antonio Parziale, Moises Diaz, Miguel A Ferrer, and Angelo Marcelli. Sm-dtw : Stability modulated dynamic time warping for signature verification. *Pattern Recognition Letters*, 121 :113–122, 2019.

[119] Luca Pasa and Alessandro Sperduti. Pre-training of recurrent neural networks via linear autoencoders. *Advances in Neural Information Processing Systems*, 27, 2014.

[120] Andrea Pascucci. Kolmogorov equations in physics and in finance. In *Elliptic and Parabolic Problems : A Special Tribute to the Work of Haim Brezis*, pages 353–364. Springer, 2005.

[121] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit : Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers*, pages 40–44. IEEE, 1993.

[122] Bhushan V Patil and Punam R Patil. An efficient dtw algorithm for online signature verification. In *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, pages 1–5. IEEE, 2018.

[123] Thomas Penzel, Jan W Kantelhardt, Chung-Chang Lo, Karlheinz Voigt, and Claus Vogelmeier. Dynamics of heart rate and sleep stages in normals and patients with sleep apnea. *Neuropsychopharmacology*, 28(1) :S48–S53, 2003.

[124] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[125] Mathias Perslev, Sune Darkner, Lykke Kempfner, Miki Nikolic, Poul Jørgen Jennum, and Christian Igel. U-sleep : resilient high-frequency sleep staging. *NPJ digital medicine*, 4(1) :72, 2021.

[126] Huy Phan and Kaare Mikkelsen. Automatic sleep staging of eeg signals : recent

development, challenges, and future directions. *Physiological Measurement*, 2022.

[127] Huy Phan, Fernando Andreotti, Navin Cooray, Oliver Y Chén, and Maarten De Vos. Automatic sleep stage classification using single-channel eeg : Learning sequential features with attention-based recurrent neural networks. In *2018 40th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 1452–1455. IEEE, 2018.

[128] Huy Phan, Fernando Andreotti, Navin Cooray, Oliver Y Chén, and Maarten De Vos. Dnn filter bank improves 1-max pooling cnn for single-channel eeg automatic sleep stage classification. In *2018 40th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 453–456. IEEE, 2018.

[129] Huy Phan, Fernando Andreotti, Navin Cooray, Oliver Y Chén, and Maarten De Vos. Seqsleepnet : end-to-end hierarchical recurrent neural network for sequence-to-sequence automatic sleep staging. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(3) :400–410, 2019.

[130] Huy Phan, Oliver Y Chén, Minh C Tran, Philipp Koch, Alfred Mertins, and Maarten De Vos. Xsleepnet : Multi-view sequential model for automatic sleep staging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44 (9) :5903–5915, 2021.

[131] Huy Phan, Kaare Mikkelsen, Oliver Y Chén, Philipp Koch, Alfred Mertins, and Maarten De Vos. Sleeptransformer : Automatic sleep staging with interpretability and uncertainty quantification. *IEEE Transactions on Biomedical Engineering*, 69(8) :2456–2467, 2022.

[132] Stuart F Quan, Barbara V Howard, Conrad Iber, James P Kiley, F Javier Nieto, George T O'Connor, David M Rapoport, Susan Redline, John Robbins, Jonathan M Samet, et al. The sleep heart health study : design, rationale, and methods. *Sleep*, 20(12) :1077–1085, 1997.

[133] Brian P Quinn. Diagnostic and statistical manual of mental disorders, primary care version. *Primary Care Companion to the Journal of Clinical Psychiatry*, 1 (2) :54, 1999.

[134] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270, 2012.

[135] K Ranjeet, A Kumar, and Rajesh K Pandey. Ecg signal compression using different techniques. In *International Conference on Advances in Computing, Communication and Control*, pages 231–241. Springer, 2011.

[136] Mauricio Reyes, Raphael Meier, Sérgio Pereira, Carlos A Silva, Fried-Michael Dahlweid, Hendrik von Tengg-Kobligk, Ronald M Summers, and Roland Wiest. On the interpretability of artificial intelligence in radiology : challenges and opportunities. *Radiology : artificial intelligence*, 2(3) :e190043, 2020.

[137] Mateus Dias Ribeiro, Abdul Rehman, Sheraz Ahmed, and Andreas Dengel. Deepcfd : Efficient steady-state laminar flow approximation with deep convolutional neural networks. *arXiv preprint arXiv:2004.08826*, 2020.

[138] Cédric Rommel, Joseph Paillard, Thomas Moreau, and Alexandre Gramfort. Data augmentation for learning predictive models on eeg : a systematic comparison. *Journal of Neural Engineering*, 19(6) :066020, 2022.

[139] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net : Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015 : 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[140] Alexandre Rouen, Jonathan Taïeb, Gabriela Caetano, Victor Pitron, Maxime Elbaz, Dominique Salmon, and Damien Leger. Polysomnographic parameters in long-covid chronic insomnia patients. *Dialogues in Clinical Neuroscience*, 25(1) : 43–49, 2023.

[141] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[142] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115 :211–252, 2015.

[143] Matthew Russell and Peng Wang. Physics-informed deep learning for signal compression and reconstruction of big data in industrial condition monitoring. *Mechanical Systems and Signal Processing*, 168 :108709, 2022.

[144] Alaa Sagheer and Mostafa Kotb. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports*, 9(1) :1–16, 2019.

[145] Ahmed Ben Said, Amr Mohamed, Tarek Elfouly, Khaled Harras, and Z Jane Wang. Multimodal deep learning approach for joint eeg-emg data compression and classification. In *2017 IEEE wireless communications and networking conference (WCNC)*, pages 1–6. IEEE, 2017.

[146] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal*

processing, 26(1) :43–49, 1978.

[147] Yasushi Sakurai, Masatoshi Yoshikawa, and Christos Faloutsos. Ftw : fast similarity search under the time warping distance. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 326–337, 2005.

[148] Tim Salimans and Durk P Kingma. Weight normalization : A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[149] David Salomon. *Data compression*. Springer, 2002.

[150] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5) :561–580, 2007.

[151] Michael J Sateia. International classification of sleep disorders. *Chest*, 146(5) : 1387–1394, 2014.

[152] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[153] Hogeon Seo, Seunghyeok Back, Seongju Lee, Deokhwan Park, Tae Kim, and Kyoobin Lee. Intra-and inter-epoch temporal context network (iitnet) using sub-epoch features for automatic sleep scoring on raw single-channel eeg. *Biomedical signal processing and control*, 61 :102037, 2020.

[154] Skyler Seto, Wenyu Zhang, and Yichen Zhou. Multivariate time series classification using dynamic time warping template selection for human activity recognition. In *2015 IEEE symposium series on computational intelligence*, pages 1399–1406. IEEE, 2015.

[155] Mark J Shensa et al. The discrete wavelet transform : wedding the a trous and mallat algorithms. *IEEE Transactions on signal processing*, 40(10) :2464–2482, 1992.

[156] Deepak Shrivastava, Syung Jung, Mohsen Saadat, Roopa Sirohi, and Keri Crewson. How to interpret the results of a sleep study. *Journal of community hospital internal medicine perspectives*, 4(5) :24983, 2014.

[157] Padhraic Smyth. Clustering sequences with hidden markov models. *Advances in neural information processing systems*, 9, 1996.

[158] Joan B Soriano, Srinivas Murthy, John C Marshall, Pryanka Relan, and Janet V Diaz. A clinical case definition of post-covid-19 condition by a delphi consensus. *The Lancet Infectious Diseases*, 22(4) :e102–e107, 2022.

[159] Arnaud Sors, Stéphane Bonnet, Sébastien Mirek, Laurent Vercueil, and Jean-François Payen. A convolutional neural network for sleep stage scoring from raw single-channel eeg. *Biomedical Signal Processing and Control*, 2017.

[160] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958, 2014.

[161] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. *IEEE transactions on Knowledge and Data Engineering*, 25(6) :1425–1438, 2012.

[162] Gregor Stiglic, Primoz Kocbek, Nino Fijacko, Marinka Zitnik, Katrien Verbert, and Leona Cilar. Interpretability of machine learning-based prediction models in healthcare. *Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery*, 10(5) :e1379, 2020.

[163] Gilbert Strang. The discrete cosine transform. *SIAM review*, 41(1) :135–147, 1999.

[164] Akara Supratak and Yike Guo. Tinysleepnet : An efficient deep learning model for sleep stage scoring based on raw single-channel eeg. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 641–644. IEEE, 2020.

[165] Akara Supratak, Hao Dong, Chao Wu, and Yike Guo. Deepsleepnet : A model for automatic sleep stage scoring based on raw single-channel eeg. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(11) :1998–2008, 2017.

[166] Adam Szczepański, Khalid Saeed, and Alois Ferscha. A new method for ecg signal feature extraction. In *Computer Vision and Graphics : International Conference, ICCVG 2010, Warsaw, Poland, September 20-22, 2010, Proceedings, Part II*, pages 334–341. Springer, 2010.

[167] Chang Wei Tan, François Petitjean, Eamonn Keogh, and Geoffrey I Webb. Time series classification for varying length series. *arXiv preprint arXiv:1910.04341*, 2019.

[168] Romain Tavenard. Differentiability of dtw and the case of soft-dtw. https://rtavenar.github.io/blog/softdtw.html, 2021.

[169] Markus Thill, Sina Däubener, Wolfgang Konen, Thomas Bäck, P Barancikova, M Holena, T Horvat, M Pleva, and R Rosa. Anomaly detection in electrocardiogram readings with stacked lstm networks. In *ITAT*, pages 17–25, 2019.

[170] Markus Thill, Wolfgang Konen, Hao Wang, and Thomas Bäck. Temporal convolutional autoencoder for unsupervised anomaly detection in time series. *Applied*

Soft Computing, 112 :107751, 2021.

[171] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 648–656, 2015.

[172] Spyros Tsevas and Dimitris K Iakovidis. Dynamic time warping fusion for the retrieval of similar patient cases represented by multimodal time-series medical data. In *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine*, pages 1–4. IEEE, 2010.

[173] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet : A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125, 2016.

[174] Jeroen Van Der Donckt, Jonas Van Der Donckt, Emiel Deprost, Nicolas Vandenbussche, Michael Rademaker, Gilles Vandewiele, and Sofie Van Hoecke. Do not sleep on traditional machine learning : Simple and interpretable techniques are competitive to deep learning for sleep scoring. *Biomedical Signal Processing and Control*, 81 :104429, 2023.

[175] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53 :5929–5955, 2020.

[176] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[177] Taras K Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1) :52–57, 1968.

[178] Zitong Wan, Rui Yang, Mengjie Huang, Nianyin Zeng, and Xiaohui Liu. A review on transfer learning in eeg signal analysis. *Neurocomputing*, 421 :1–14, 2021.

[179] Fei Wang, Qiming Ma, Wenhan Liu, Sheng Chang, Hao Wang, Jin He, and Qijun Huang. A novel ecg signal compression method using spindle convolutional autoencoder. *Computer methods and programs in biomedicine*, 175 :139–150, 2019.

[180] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26 :275–309, 2013.

[181] Renjie Wu and Eamonn J Keogh. Fastdtw is approximate and generally slower than the algorithm it approximates. *IEEE Transactions on Knowledge and Data*

Engineering, 34(8) :3779–3785, 2020.

[182] Lexiang Ye and Eamonn Keogh. Time series shapelets : a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956, 2009.

[183] Ozal Yildirim, Ru San Tan, and U Rajendra Acharya. An efficient compression of ecg signals using deep convolutional autoencoders. *Cognitive Systems Research*, 52 :198–211, 2018.

[184] Ozal Yildirim, Ulas Baran Baloglu, and U Rajendra Acharya. A deep learning model for automated sleep stages classification using psg signals. *International journal of environmental research and public health*, 16(4) :599, 2019.

[185] Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. Embedding with autoencoder regularization. In *Machine Learning and Knowledge Discovery in Databases : European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*, pages 208–223. Springer, 2013.

[186] Guo-Qiang Zhang, Licong Cui, Remo Mueller, Shiqiang Tao, Matthew Kim, Michael Rueschman, Sara Mariani, Daniel Mobley, and Susan Redline. The national sleep research resource : towards a sleep data commons. *Journal of the American Medical Informatics Association*, 25(10) :1351–1358, 2018.

[187] Yaodong Zhang and James R Glass. An inner-product lower-bound estimate for dynamic time warping. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5660–5663. IEEE, 2011.

[188] Dechun Zhao, Renpin Jiang, Mingyang Feng, Jiaxin Yang, Yi Wang, Xiaorong Hou, and Xing Wang. A deep learning algorithm based on 1d cnn-lstm for automatic sleep staging. *Technology and Health Care*, 30(2) :323–336, 2022.

[189] Zhong Zheng, Zijun Zhang, Long Wang, and Xiong Luo. Denoising temporal convolutional recurrent autoencoders for time series classification. *Information Sciences*, 588 :159–173, 2022.

[190] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1) :43–76, 2020.