UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE – Faculté des sciences

MÉMOIRE

Présenté en vue d'obtenir

MASTER SEP

Classification des séries temporelles par réseaux de neurones LSTM

Maxime ANGOULVENT

Sous la direction de : **M. KEZIOU Amor, Maître de conférences – HDR**

JURY

MEMBRES : M. Jules MAES

M. Amor KEZIOU

Soutenu le 28/05/2021

Remerciements

Je tiens à remercier toutes les personnes ayant contribué à la conception de ce mémoire.

Je voudrais tout d'abord remercier Madame GAUTHERAT, directrice du master SEP à Reims pour m'avoir proposé ce sujet.

J'aimerais aussi remercier mon directeur de mémoire M. KEZIOU pour son implication et ses conseils.

Je souhaite également remercier les professeurs de l'université de Reims Champagne Ardenne pour m'avoir donné les moyens de réussir mes études.

Je désire aussi remercier mon camarade Ouael ETTOUILEB pour m'avoir aidé à trouver une base de données ainsi que de m'avoir éclairé sur certains points.

Un grand merci à mademoiselle Juliette ABLINE pour son travail de relecture et ses critiques constructives concernant la compréhension de certaines phrases.

Résumé

Avec l'avancée technologique, collecter des données n'a jamais été aussi simple. Les réseaux de neurones artificiels sont inspirés des réseaux de neurones biologiques et nécessitent de grandes bases de données. Ainsi, le Deep Learning devient de plus en plus populaire. Nous distinguons 2 types de réseau de neurones artificiels, les réseaux feed-forward et les réseaux récurrents. L'information ne circule que dans un seul sens pour les réseaux feed-forward alors qu'elle peut se propager dans les deux sens pour les réseaux récurrents. Ces réseaux récurrents sont utiles pour traiter des séries temporelles. Les réseaux de neurones LSTM sont des réseaux récurrents qui ont la particularité d'avoir une mémoire à long terme pour traiter des séquences plus longues. Ces réseaux peuvent également être utilisés pour un travail de classification. Pour cela, il faut rajouter des couches denses aux couches LSTM du réseau. Nous avons créé 3 modèles pour classifier l'activité humaine à partir de données (x, y, z d'un accéléromètre) récoltées dans le temps. Le modèle LSTM est le plus efficace pour prédire l'activité car il comprend le lien temporel dans la récolte des données qu'un réseau classique ne détectera pas.

Mots-clefs : réseau de neurones, Deep Learning, réseau feed-forward, réseau récurrent, réseau LSTM, classification, réseau profond, réseau convolutif, rétropropagation, séries temporelles.

Abstract

With technological advances, collecting data has never been easier. Artificial neural networks are inspired by biological neural networks and require large databases. As a result, Deep Learning is becoming more and more popular. We distinguish 2 types of artificial neural networks, feed-forward networks and recurring networks. Information flows only in one direction for feed-forward networks whereas it can propagate in both directions for recurring networks. These recurring networks are useful for processing time series. LSTM neural networks are recurrent networks that have the particularity of having a long-term memory to treat longer sequences. These networks may also be used for classification work. In order to do this, dense layers must be added to the LSTM layers of the network. We created 3 models to classify human activity from data (x, y, z of an accelerometer) collected over time. The LSTM model is the most effective in predicting activity because it understands the temporal link in the collection of data that a conventional network will not detect.

Keywords : neural network, Deep Learning, feed-forward network, recurring network, LSTM network, classification, deep network, convolutive network, backscatter, time series.

Liste des abréviations

CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
m	Mètre
mm	Millimètre
ms	Milliseconde
mv	Millivolt
RELU	Rectified Linear units
RNA	Réseaux de neurones artificiels
RNN	Recurrent Neural Network
SNC	Système nerveux central

Glossaire

Classification	**Processus de regroupement des données dans des catégories prédéfinies.**
Concaténation	**Processus mathématique permettant de regrouper des matrices ou des vecteurs entre eux.**
Deep Learning	**Le Deep Learning est également appelé apprentissage profond. Il repose sur le traitement de grandes quantités de données à l'aide de réseaux de neurones artificiels.**
Dense	**Une couche est dit dense si tous ces neurones sont liés aux neurones de la couche précédente.**
Données d'entrainement	**Données sur lesquelles notre réseau de neurones va s'entrainer pour devenir de plus en plus efficace.**
Données de test	**Données mises en réserve que notre réseau de neurones n'a jamais vues. Ces données vont nous servir pour vérifier si notre réseau de neurones est réellement efficace.**
Epoque	**Nombre de fois que les données d'entrainement vont être répétées dans notre réseau de neurones.**
Gaine de myéline	**La myéline est composée majoritairement de lipides. La gaine de myéline permet d'augmenter la vitesse de l'influx nerveux.**
Réseau de neurones profonds	**Un réseau de neurones est dit profond s'il est composé d'au moins 3 couches cachées et d'une couche de sortie.**
Rétropropagation	**Méthode mathématique qui permet de modifier les poids et les biais du réseau de neurones.**

SOMMAIRE

INTRODUCTION

Les réseaux de neurones artificiels sont en expansion depuis une dizaine d'années. Ils sont aujourd'hui présents dans de nombreux secteurs d'activité tel que la médecine, la finance. Ces réseaux sont utilisés pour la reconnaissance vocale et d'image, pour la prédiction de cours boursiers et bien d'autres.

Le Deep Learning connaît une grande expansion. Voulant me diriger vers le métier de data analyst, data scientist, il était nécessaire de s'intéresser aux réseaux de neurones et à leur fonctionnement.

De nos jours, les téléphones sont omniprésents et leur technologie ne fait qu'augmenter. Il est de plus en plus facile de recueillir des informations sur leurs propriétaires.

Dans ce mémoire, nous nous intéresserons aux données recueillies à l'aide d'un accéléromètre intégré dans les téléphones. Nous nous interrogerons sur le fonctionnement des réseaux de neurones LSTM pour classifier des séries temporelles (données x, y, z de l'accéléromètre) en différentes activités (marche, course, …). Nous comparerons nos résultats avec d'autres réseaux pour vérifier la fiabilité des prédictions obtenues.

La première partie a pour vocation d'expliquer les neurones biologiques, leur structure, leur fonctionnement, les différents types existant ainsi que leur rôle. La seconde partie répondra aux questions suivantes : Qu'est-ce qu'un réseau de neurones artificiels ? Comment un réseau de neurones artificiels apprend ? Quels sont les différents réseaux de neurones artificiels et quels sont leurs spécificités ? Dans la troisième partie, nous discuterons des résultats des prédictions obtenus lors de notre codage des réseaux de neurones profonds, convolutifs et LSTM.

PARTIE I
LE NEURONE CHEZ LE MAMMIFERE

1. La structure d'un neurone

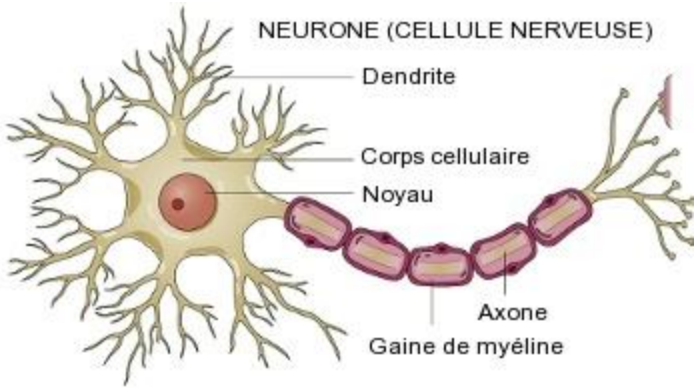

Figure 1 Structure d'un neurone[1]

Le neurone est une cellule[1] située dans le système nerveux central. On l'appelle également cellule nerveuse (voir figure 1). Le bon fonctionnement du cerveau repose sur l'interconnexion des neurones[2]. Le nombre de neurones varie en fonction des espèces pouvant aller de 100 millions à 100 milliards de neurones. Les neurones sont composés de plusieurs parties, le soma, les dendrites et l'axone. Nous allons d'abord parler du soma. Celui-ci est le corps cellulaire du neurone, il contient la membrane plasmique, le cytoplasme ainsi que les organites baignant à l'intérieur comme le noyau.

Les dendrites sont courtes et nombreuses. Ce sont des prolongements du corps cellulaire. Elles se ramifient et forment l'arbre dendritique (voir figure 2).[3]

Tout comme les dendrites, l'axone est également un prolongement du soma. Il peut mesurer de 1 mm à 1 m chez l'homme[4]. Il est composé d'une membrane et d'un cytoplasme. Tous les axones sont entourés des cellules de Schwann qui une fois enroulées sur elles-mêmes forment des gaines de myéline.

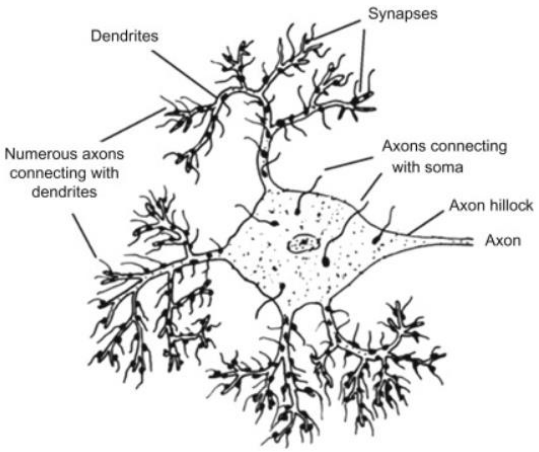

Figure 2 L'arbre dendritique[5]

[1] Alliance, « La cellule nerveuse (neurone) ».
[2] « Le neurone ».
[3] « Dendrites- un aperçu | Sujets ScienceDirect ».
[4] « Structure- L'axone ».
[5] « Dendrites- un aperçu | Sujets ScienceDirect ».

2. Le fonctionnement d'un neurone

Les neurones reçoivent et transmettent des informations avec d'autres neurones. On parle alors de réseaux de neurones. L'information est recueillie au niveau des dendrites[6] et va ensuite être acheminée vers le soma. C'est l'axone qui conduit l'information du corps cellulaire jusqu'à la synapse. L'information à l'intérieur du neurone est électrique : nous pouvons l'enregistrer à l'aide d'un voltmètre [7]. Nous parlons de potentiel d'action lorsque l'intensité à l'intérieur de l'axone varie d'un seuil à l'autre, d'environ -50 mv, -70mv à plus de 30 mv (voir figure 3). Cela prend environ 1ms et correspond à l'échange du message nerveux.

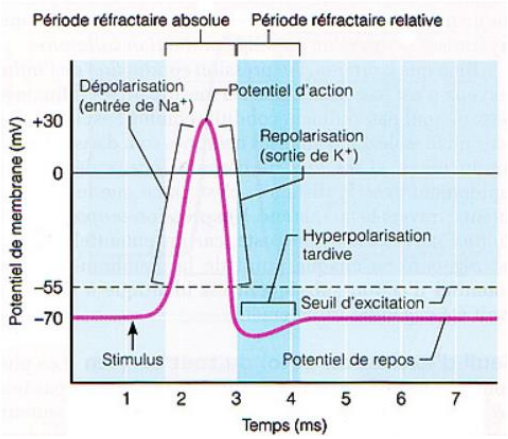

Figure 3 Potentiel d'action[8]

Comme dit précédemment, certains neurones sont recouverts de gaines de myéline qui permettent d'accélérer la vitesse du message. L'échange d'informations entre deux neurones s'effectue au niveau des synapses[9]. Les neurones ne se touchent pas pour se transmettre l'information : on parle d'espace synaptique. Dans cet espace, les signaux électriques ne circulent pas, les neurotransmetteurs prennent le relai (voir figure 4). Ce sont des composés chimiques, on parle donc de message électrochimique. Ils vont se déposer sur des récepteurs au niveau de la membrane post-synaptique.

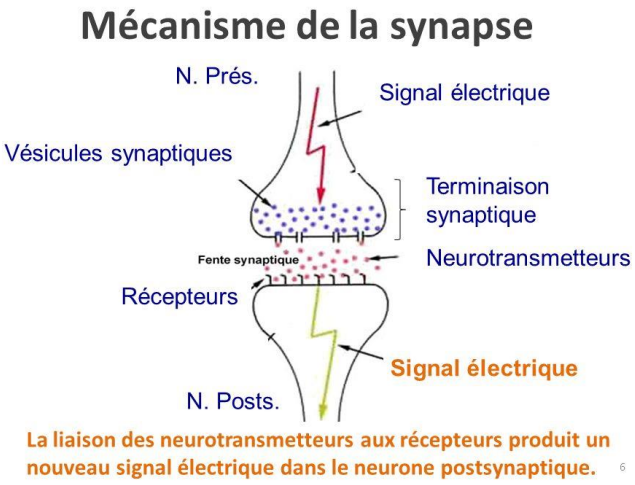

Figure 4 Echange d'informations au niveau de la synapse[10]

[6] *L'explication la plus facile - Neurones et Transmission Neuronale.*
[7] Corpus : Au cœur des organes. Le système nerveux
[8] « Le Neurone- cellule fondamentale du syst🅾me nerveux. »
[9] « La communication entre les neurones ».
[10] « D4. Mécanisme neuronal de transmission de l'information- ppt télécharger », 4.

3. Les différents types de neurones

Il existe plus de 200 types de neurones différents[11]. Chaque neurone a une place particulière dans le système nerveux, une forme particulière et des connexions variées avec d'autres neurones ou des cellules. Cela explique les différents types de neurones répertoriés. Nous pouvons les classer en deux catégories :

- D'un point de vue fonctionnel
- D'un point de vue morphologique

Chaque catégorie se divise en 3 parties.

3.1. Point de vue fonctionnel

D'un point de vue fonctionnel, nous retrouvons les neurones sensoriels, les interneurones et les neurones moteurs (figure 5).

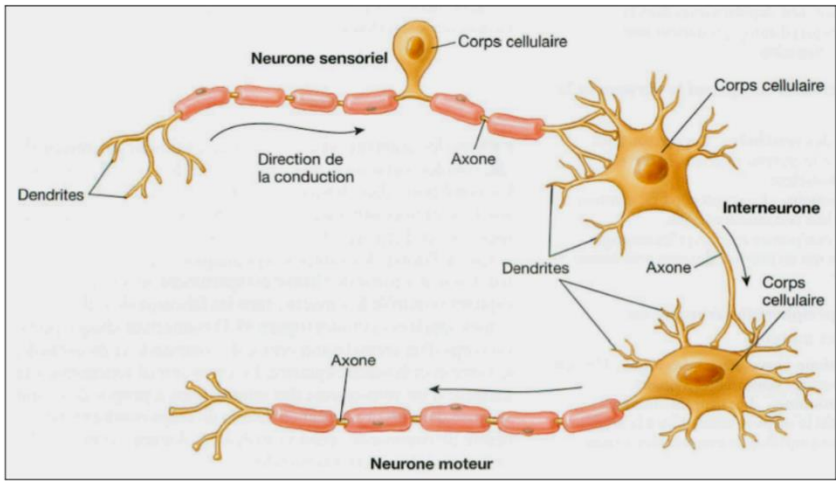

Figure 5 Les neurones d'un point de vue fonctionnel[12]

3.1.1. Les neurones sensoriels

Les neurones sensoriels captent les messages des récepteurs sensoriels grâce à leurs dendrites. Ces neurones transmettent l'information dans le système nerveux central (SNC) au niveau du cerveau et de la moelle épinière (figure 6).

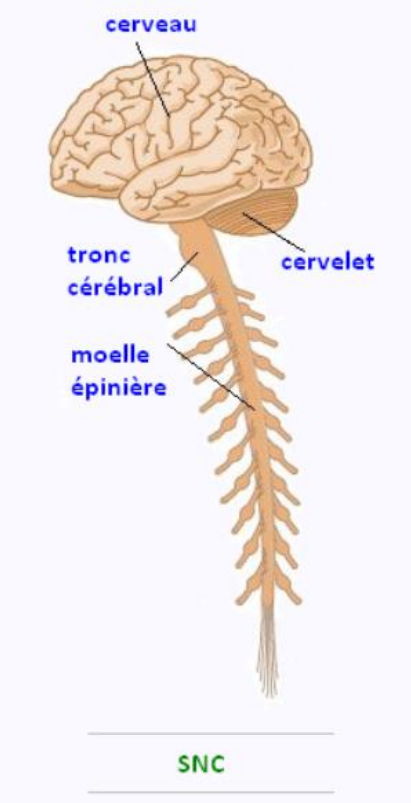

Figure 6 Système nerveux central [13]

[11] « LE CERVEAU À TOUS LES NIVEAUX! »
[12] « Les cellules nerveuses et la moelle épinière ».
[13] « Sciences et technologies ».

3.1.2. Les interneurones

Les interneurones sont situés dans le cerveau et la moelle épinière. Ils connectent différents neurones et permettent l'association de l'activité sensorielle et motrice. [14]

3.1.3. Les neurones moteurs

Les neurones moteurs sont également appelés « motoneurones ». Ce sont des cellules nerveuses. Ils sont connectés aux muscles et commandent leur contraction.

3.2. Point de vue morphologique

D'un point de vue morphologique, nous retrouvons les neurones pseudo-unipolaires (figure 7), les neurones multipolaires (figure 8) et les neurones bipolaires (figure 9).

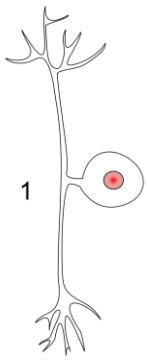

Figure 7 Neurone pseudo-unipolaire [15]

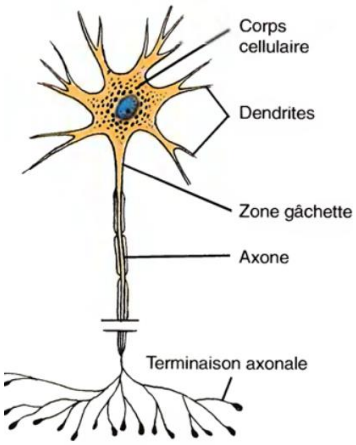

Figure 8 Neurone multipolaire[16]

Figure 9 Neurone bipolaires[17]

Les neurones pseudo-unipolaires ont leur axone divisé en 2 parties distinctes.

Les neurones multipolaires ont de nombreuses dendrites relativement courtes. Ils possèdent un seul axone de longue taille.

Les neurones bipolaires sont divisés en 2 prolongements, l'un d'axone et l'autre de dendrites.

[14] Usito, « Usito ».
[15] « Neurone pseudo-unipolaire ».
[16] « Système nerveux central | Système nerveux, Système nerveux central, Physiologie ».
[17] « Polarisation des neurones ».

4. Quel est le rôle des neurones

Les neurones reçoivent des informations et en transmettent à d'autres neurones ou cellules. Ces cellules peuvent être musculaires par exemple. Chaque neurone a un type et une morphologie précise dû à son emplacement dans le SNC ainsi que les liaisons qu'il effectue. Chaque neurone est donc différent et a donc des fonctions différentes. On peut noter l'apprentissage, la motricité, la posture, etc…

PARTIE II
LES RESEAUX DE NEURONES ARTIFICIELS

1. Qu'est ce que les réseaux de neurones artificiels

Le terme « réseaux de neurones artificiels » est apparu en 1943. Il a été pensé par Warren McCullough un neurophysicien et Walter Pitts un mathématicien.[18]

Les RNA (réseaux de neurones artificiels) ont besoin de machines suffisamment puissantes pour effectuer des listes de calculs ainsi que de larges bases de données. C'est pour cela qu'il a fallu attendre jusqu'en 2010 pour que les RNA soient véritablement exploités.

En 2012, un RNA a accompli ses premières prouesses. Il a réussi dans le cadre de reconnaissance d'images à être plus efficace qu'un humain.

Actuellement, les RNA sont utilisés dans différents domaines tels que les mathématiques, la médecine, l'économie, etc... [19]

Les RNA sont très différents d'un algorithme « classique ». En effet, ils ne vont pas procéder de la même manière pour obtenir un résultat. Lorsque l'on code un algorithme, il va falloir lui expliquer toutes les étapes en détail. Au contraire, le RNA apprend par lui-même. Il est basé sur les réseaux de neurones biologiques. Cela peut ainsi faire penser à un cerveau d'enfant qui va apprendre de ses erreurs pour ne plus les reproduire.

1.1. Comment fonctionne le réseau de neurones artificiels

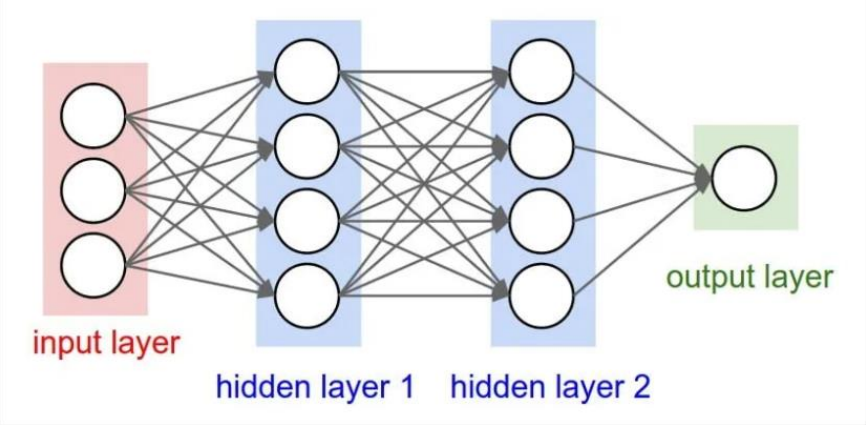

Figure 10 Schéma des Réseaux de neurones artificiels [20]

Les RNA sont divisés en plusieurs couches. La première est l'input layer, couche d'entrée qui contient les informations et va les transmettre à la seconde couche, hidden layer 1 ou « couche cachée 1 ».

[18] L, « Réseau de neurones artificiels ».
[19] Schmitt et al., « Les réseaux de neurones artificiels. Un outil de traitement de données prometteur pour l'anthropologie ».
[20] L, « Réseau de neurones artificiels ».

Cette transmission va se faire le long des flèches[21] sur le schéma (figure 10) que l'on appelle synapse en référence au terme biologique. Chaque synapse peut être inhibitrice ou activatrice à un degré d'intensité variable (peu, moyennement, beaucoup). De plus, un signal avec sa propre intensité va circuler le long le de la synapse. Prenons xi et wi des nombres réels, avec xi l'intensité du signal qui est positive et wi, un poids qui est négatif si la synapse est inhibitrice et positif si la synapse est activatrice. L'intensité de la synapse et du signal vont se multiplier. Ainsi, un neurone va recevoir une somme de ces multiplications linéaires de la couche précédente plus un biais. Mathématiquement, cela s'observe de la forme $w_1x_1 + w_2x_2 + \cdots + w_Nx_N + biais$. A ceci s'ajoute une fonction d'activation que l'on peut noter σ.

On obtient donc $\sigma(w_1x_1 + w_2x_2 + \cdots + w_Nx_N + biais)$. Nous pouvons procéder sous forme de matrice. Notons a_1^1, a_2^1, …, a_n^1 les neurones de la première couche et a_1^2, a_2^2, …, a_k^2 les neurones de la seconde couche. Nous obtenons alors :

$$\begin{bmatrix} a_1^2 \\ a_2^2 \\ \vdots \\ a_k^2 \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{n,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,k} & w_{2,k} & \cdots & w_{n,k} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \right)$$

En fonction du résultat, le neurone va pouvoir s'activer ou non. Le neurone s'active si le résultat se situe entre 0 et 1. Un résultat de 0,5 signifie que le neurone sera activé à 50 pourcents. Le biais permet de modifier cette valeur. Il va décaler la fonction d'activation.

Cela se poursuit de couche en couche jusqu'à atteindre l'output layer ou couche de sortie.

1.2. Comment le réseau de neurones artificiels apprend

Prenons l'exemple d'une reconnaissance de chiffres variant entre 0 et 9. [22] Chaque chiffre sera représenté sur une image en noir et blanc de 28 pixels par 28. Le chiffre est représenté en blanc et le reste de l'image en noir. Nous obtenons donc au total 784 pixels (28x28) prenant la valeur 0 si le pixel est situé sur une case noire et 1 si le pixel est situé sur une case blanche.

Si nous demandons à un groupe de 100 personnes de nous écrire le chiffre 3 de manière manuscrite, nous obtiendrons une centaine d'écritures différentes pour un même chiffre. Par exemple, la courbe du 3 ne sera pas forcément la même pour tout le monde. Pourtant, le RNA devra reconnaitre tous ces chiffres comme étant le chiffre 3. La difficulté est que chaque chiffre n'est pas écrit de la même manière.

Pour commencer, nous allons donc prendre 784 neurones d'entrées et 10 neurones de sorties. Les neurones d'entrées correspondent au nombre de pixels et les neurones de sorties au nombre de chiffres que l'on peut obtenir. Si l'on donne à l'ordinateur une image du chiffre 5, nous nous attendons

[21] Science4All, *Les réseaux de neurones | Intelligence artificielle 41.*
[22] Chronophage, *Les maths des réseaux de neurones - Introduction au Deep Learning #2.*

à ce que seul le neurone de sortie correspondant à ce chiffre s'active donc qu'il renvoie la valeur 1. Or, ce n'est pas le cas immédiatement. Lors du premier essai, tous les neurones de sorties vont s'activer plus ou moins fortement. Le RNA n'a pas réussi à reconnaître l'image. Ces erreurs sont dues au fait que les valeurs des poids et des biais sont toujours aléatoires au premier essai. Il va falloir calculer le coût.

Il est fréquemment calculé avec la formule [23] $C=1/n \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ où n est le nombre de sorties possibles, ici 10, y_i correspond à la valeur affichée par le neurone de sortie et $\hat{y}_i$ correspond à la valeur attendue (0 si le neurone ne devait pas s'activer, 1 sinon).

Il ne reste plus qu'à modifier les poids et les biais afin d'obtenir le coût le plus faible possible. Pour cela, nous allons utiliser la méthode de rétropropagation du gradient[24]. Cette méthode permet de minimiser la fonction de coût en dérivant jusqu'à ce que la pente en un point soit égale à 0. Les poids se modifient avec la formule w=w-$\eta \frac{\partial C_0}{\partial w^{(L)}}$ où η est le taux d'apprentissage. Le biais se modifie avec la formule b=b-$\eta \frac{\partial C_0}{\partial b^{(L)}}$. Une valeur élevée de η permet une convergence rapide vers le minimum de la fonction mais elle sera moins précise qu'avec une valeur faible.

Nous allons expliquer cette méthode dans le cas d'un réseau de neurones simples avec une seule entrée, un neurone de sortie et 2 couches cachées comportant chacune un neurone. Notons $A^{(L)}$ l'activation du neurone de sortie et $A^{(L-1)}$ l'activation du neurone précédent. Comme nous l'avons vu précédemment : $A^{(L)}$=σ($w^{(L)}A^{(L-1)}$+$b^{(L)}$). Notons $z^{(L)} = w^{(L)}A^{(L-1)}$+$b^{(L)}$, nous pouvons donc dire que $A^{(L)} =$ σ($z^{(L)}$). Maintenant que nous savons comment modifier les poids, il nous faut calculer $\frac{\partial C_0}{\partial w^{(L)}}$ =$\frac{\partial C_0}{\partial A^{(L)}}\frac{\partial A^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial w^{(L)}}$ pour modifier la valeur du poids $w^{(L)}$. Cette valeur correspond à combien le poids $w^{(L)}$fait varier le coût.

Nous allons maintenant nous intéresser à la valeur de chacune des dérivées composant le calcul :

- $\frac{\partial C_0}{\partial A^{(L)}} = 2\left(A^{(L)} - y\right)$
- $\frac{\partial A^{(L)}}{\partial z^{(L)}} = \sigma'\left(z^{(L)}\right)$
- $\frac{\partial z^{(L)}}{\partial w^{(L)}} = A^{(L-1)}$

Nous obtenons donc $\frac{\partial C_0}{\partial w^{(L)}} = 2\left(A^{(L)} - y\right)\sigma'\left(z^{(L)}\right)A^{(L-1)}$. Nous remarquons que la dernière dérivée dépend de l'activation du neurone précédent ($A^{(L-1)}$). Cela signifie que la modification du poids $w^{(L)}$ dépend du neurone antérieur. Il va donc être important de modifier les poids et les biais des neurones précédents si nous voulons avoir le coût le plus faible possible. Dans cet exemple, nous effectuons ce calcul pour un seul entrainement, cependant, dans la réalité, les données d'entrainement sont répétées dans le réseau de neurones autant de fois qu'on le souhaite. Chaque répétition supplémentaire correspond à une époque. Cela signifie que tous ces calculs s'effectueront pour chaque poids et chaque biais à chaque fois que les données d'entrainement repasseront dans le réseau

[23] Lemaire et al., « Une nouvelle fonction de coût régularisante dans les réseaux de neurones artificiels ».
[24] *Rétropropagation de gradient, analyse. Le deep learning.*

de neurones. La modification du biais $b^{(L)}$ est très similaire à la modification du poids. Elle s'obtient par la formule :

$\frac{\partial C_0}{\partial b^{(L)}}=\frac{\partial z^{(L)}}{\partial b^{(L)}}\ \frac{\partial A^{(L)}}{\partial z^{(L)}}\ \frac{\partial C_0}{\partial A^{(L)}}=1\sigma'\left(z^{(L)}\right)2\left(A^{(L)}-y\right).$

Il ne reste plus qu'à modifier les poids et les biais des neurones précédents par la même méthode.

Nous allons maintenant voir la différence avec un réseau de neurones plus complexe (figure 11).[25]

Puisque nous sommes sur un réseau plus complexe, certaines notations vont changer. Désormais, nous noterons ω_{ij}^{L} le poids qui relie le i eme neurone de la couche L au j eme neurone de la couche L-1. Nous noterons également b_i^l le biais du i eme neurone de la couche L.

Dans cette exemple, $z_i^L=\sum_{j=1}^{n}\omega_{ij}^{L}\,A_j^{L-1}+b_i^l$.

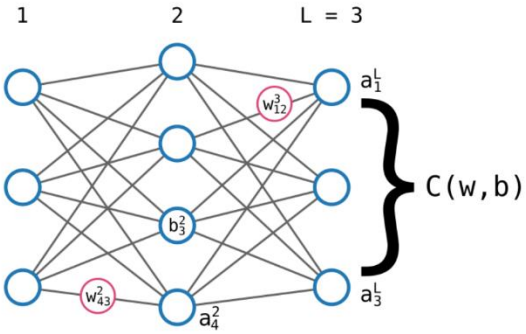

Figure 11 Réseau de neurones multicouches

La formule de modification des poids et des biais reste la même. On a donc :

- $\omega_{ij}^{L}=\omega_{ij\_}^{L}\eta\frac{\partial C}{\partial w_{ij}^{L}}$
- $b_i^l=b_i^l\_\eta\frac{\partial c}{\partial b_i^L}$

Il faut donc connaître $\frac{\partial C}{\partial w_{ij}^{L}}$ et $\frac{\partial c}{\partial b_i^L}$.

- $\frac{\partial C}{\partial w_{ij}^{L}}=\frac{\partial z_i^L}{\partial w_{ij}^{L}}\frac{\partial A_i^L}{\partial z_i^L}\frac{\partial C}{\partial A_i^L}$
- $\frac{\partial c}{\partial b_i^L}=\frac{\partial Z_i^L}{\partial b_i^L}\frac{\partial A_i^L}{\partial z_i^L}\frac{\partial C}{\partial A_i^L}$

Nous allons maintenant nous intéresser à chaque dérivée de l'équation.

- $\frac{\partial z_i^L}{\partial w_{ij}^{L}}=A_i^{L-1}$
- $\frac{\partial A_i^L}{\partial z_i^L}=\sigma'\left(z_i^L\right)$
- $\frac{\partial C}{\partial A_i^L}=2\left(A_i^L-y\right)$
- $\frac{\partial z_i^L}{\partial b_i^L}=1$

Nous obtenons les mêmes résultats de dérivées avec le réseau de neurones multicouches. La seule différence est que Z et A sont désormais des sommes ($z_i^L=\sum_{j=1}^{n}\omega_{ij}^{L}\,A_j^{L-1}+b_i^l$ et $A_i^L=\sigma\left(z_i^L\right)$).

[25] « Deep learning : la rétropropagec les ration du gradient ».

2. Les différents types de réseaux de neurones

2.1. Les réseaux de neurones feed-forward[26]

Les réseaux de neurones feed-forward sont les réseaux de neurones que nous avons vus jusqu'à présent. L'information ne circule que dans un seul sens, de l'input layer à l'output layer. Autrement dit, l'information ne se propage que de la couche d'entrée vers la couche de sortie.

Il existe deux types de réseaux de neurones feed-forward.

2.1.1. Les réseaux monocouches (perceptron simple)

Les réseaux monocouches ne disposent que d'une seule couche de neurones entre l'input layer et l'output layer. Ce réseau est idéal pour classifier des données en deux catégories. Par exemple, il peut trouver si le mail que l'on vient de recevoir est un mail que l'on peut ouvrir en toute sécurité ou si celui-ci est un spam[27]. Cette classification se fait de manière linéaire. Plusieurs fonctions d'activation sont possibles telles que la fonction de Heaviside ou la fonction sigmoïde.

2.1.2. Les réseaux multicouches (perceptron multicouche)

Les réseaux de neurones multicouches disposent de couches cachées entre l'input layer et l'output layer. Ces réseaux sont adaptés pour les fonctions non linéaires. Plusieurs fonctions d'activation sont possibles avec ces réseaux[28] : la fonction linéaire, RELU (Rectified Linear Units), sigmoïde, tangente hyperbolique.

2.1.2.1. Les réseaux neuronaux convolutifs [29]

Les réseaux neuronaux convolutifs ou CNN sont des réseaux multicouches. Ils sont basés sur le cortex visuel des animaux. Ils sont principalement utilisés pour la reconnaissance d'images et de vidéos. Ils peuvent également être utilisés dans le cas de séries temporelles. Ils sont divisés en 2 parties : la partie convolutive et la partie classification[30] [31]. La partie convolutive va grandement réduire le nombre de calculs effectués par le réseau de neurones. Différents filtres vont traiter une partie de l'information pour pouvoir gérer une information plus complexe.

[26] « Démystifier le Machine Learning, Partie 2 ».
[27] « Réseau de neurones : perceptron monocouche- IA- IAD- Java : Supports de cours ».
[28] Rakotomalala, « Perceptrons simples et multicouches ». page 28
[29] « Réseau de neurones convolutifs ».
[30] « Les réseaux de neurones convolutifs. »
[31] « Convolutional neural network | Deep Learning | DataScientest ».

2.2. Les réseaux de neurones récurrents

Les réseaux de neurones récurrents sont des réseaux de neurones où l'information peut communiquer dans les deux sens contrairement aux réseaux vus précédemment[32]. Ces réseaux stockent des informations pour pouvoir les réutiliser par la suite dans d'autres calculs. Cela signifie qu'à un instant t, le réseau utilise un certain nombre d'états passés (figure 12).

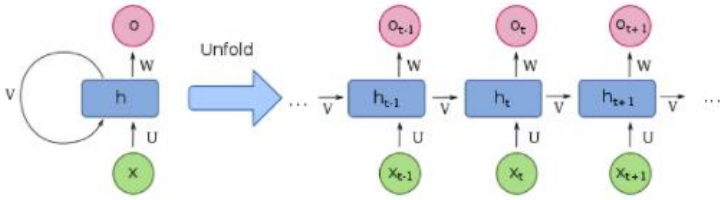

Figure 12 Schéma du réseau de neurones récurrents[33]

Ces réseaux sont également plus flexibles (figure 13). En fonction du nombre d'entrées, nous pouvons obtenir différentes sorties possibles ce qui n'était pas forcément le cas avec les réseaux vus précédemment.

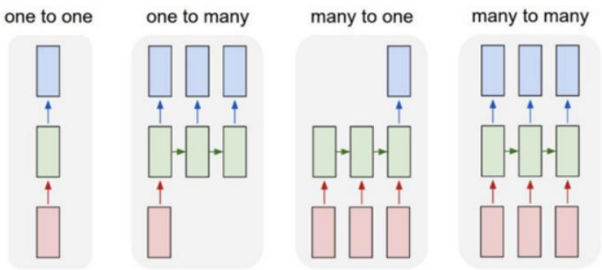

Figure 13 Différentes configurations des réseaux de neurones récurrents[34]

[32] « Réseaux de neurones récurrents ».
[33] « WISDM Lab: Wireless Sensor Data Mining ».
[34] User, « Réseaux Neuronaux Récurrents et LSTM ».

Dans cet exemple (figure 14), nous voulons que le réseau de neurones continue une phrase. Par abus de langage, nous dirons « Comment » pour exprimer le vecteur correspondant au mot « comment » et de même pour les autres mots de cet exemple. On introduit donc en entrée le mot « comment ». Le réseau de neurones nous renvoie le mot « ça ». Puis, on réintroduit le mot « ça » en entrée. Le réseau de neurones aurait une infinité de réponses à donner après ce mot. Seulement, ce réseau stocke l'information précédente pour la réutiliser. Cela veut dire que lorsque l'on introduit le mot « ça » en entrée, le réseau sait que le mot « comment » est situé avant dans la phrase et va pouvoir nous renvoyer une réponse adéquate et ainsi de suite.

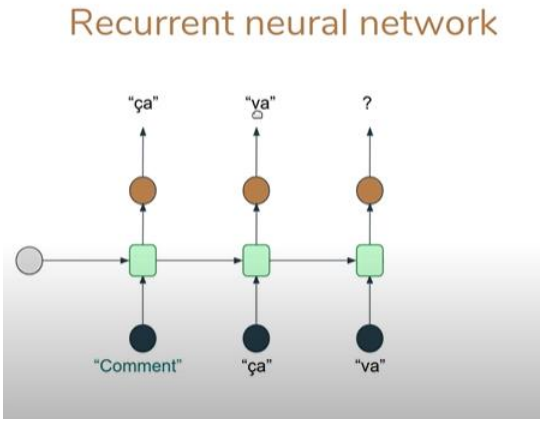

Figure 14 Réseaux de neurones récurrents[35]

Cependant, ces réseaux ont des difficultés pour des séquences trop longues. En effet, ces réseaux ont une mémoire à court terme et commencent à oublier après cinquante itérations environ.

[35] Thibault Neveu, *Comprendre les réseaux de neurones récurrents (RNN).*

3. Qu'est ce qu'un réseau de neurones LSTM

Les réseaux de neurones LSTM sont des réseaux récurrents. Cela signifie qu'ils sont très efficaces pour traiter des séries temporelles. Ces réseaux ont une mémoire à court et long terme. Cela leur permet d'éviter les problèmes de gradient des réseaux récurrents pour pouvoir traiter des séquences plus longues et plus complexes. Le réseau de neurones LSTM apprend plus rapidement qu'un réseau récurrent et a réussi ce qu'aucun réseau n'avait encore réussi à faire [36].

Comment fonctionne ce réseau ?

Les réseaux de neurones LSTM se distinguent en 3 parties : le forget gate, l'input gate et l'output gate.

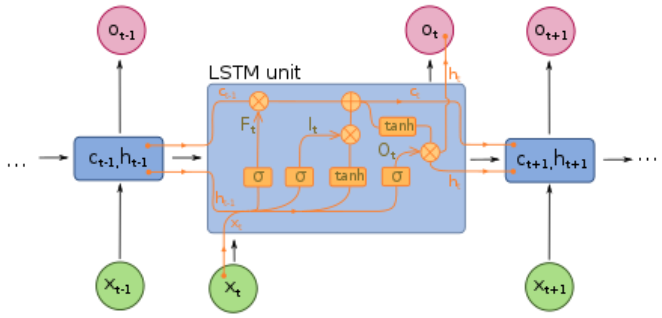

Figure 15 Schéma du réseau de neurones LSTM [37]

Mais avant d'expliquer en détail ce qui se passe, rappelons-nous du fonctionnement mathématique des réseaux de neurones expliqué plus tôt :

$$\begin{bmatrix} a_1^2 \\ a_2^2 \\ \vdots \\ a_k^2 \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{n,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,k} & w_{2,k} & \cdots & w_{n,k} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \right)$$, avec a la valeur transmisse au neurone, σ la fonction d'activation, W le poids et x l'intensité autrement dit, la valeur du vecteur d'entré et b le biais du neurone.

3.1. Forget Gate

Le forget gate a pour rôle d'oublier des informations non essentielles au fonctionnement du réseau de neurones. Cela va permettre de ne pas surcharger le réseau d'informations inutiles, contrairement aux réseaux récurrents.

Nous allons maintenant zoomer sur ce qui se passe dans le forget gate (figure 16).

[36] Hochreiter et Schmidhuber, « Long Short-Term Memory ».
[37] « Réseau de neurones récurrents ».

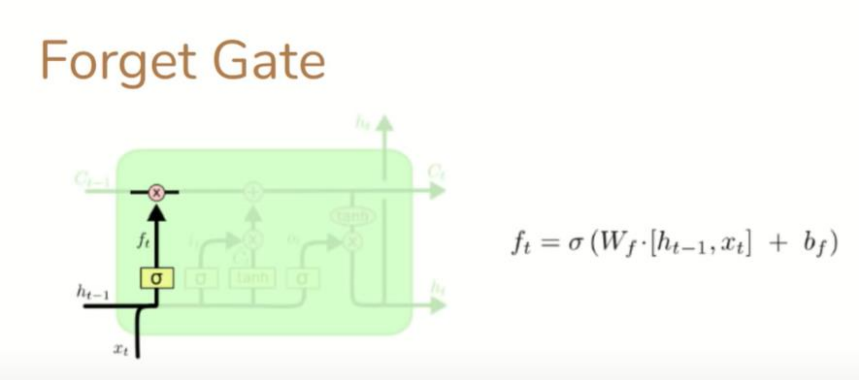

Figure 16 Zoom sur le forget gate[38]

Nous pouvons remarquer sur cette formule mathématique que la fonction d'activation, le biais, la valeur d'entrée sont bien présents. Pourtant, nous observons un terme h_{t-1} qui n'était pas présent dans les réseaux de neurones feed-forward. h_{t-1} correspond à l'état de la cellule au temps t-1.

La différence avec la formule mathématique vue précédemment est qu'au lieu de multiplier la matrice de poids par le vecteur d'entré x, nous allons multiplier cette même matrice de poids par la concaténation entre l'état de la cellule au temps t-1 et le vecteur d'entré x. Les résultats très proches de 0 seront susceptibles d'être oubliés par le réseau de neurones.

3.2. Input Gate

L'input gate a pour but de rajouter de l'information.

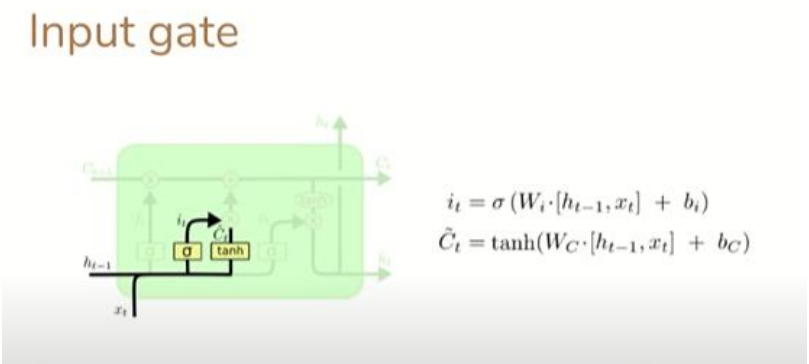

Figure 17 Zoom sur l'input gate [39]

[38] Thibault Neveu, *Comprendre les LSTM - Réseaux de neurones récurrents*.
[39] Thibault Neveu, *Comprendre les LSTM - Réseaux de neurones récurrents*, 2019.

L'input gate est divisé en 2 calculs mathématiques. Tout d'abord nous calculons les valeurs du vecteur $\widetilde{C_t}$. La fonction d'activation cette fois-ci est un tanh dont le but est de retourner des valeurs comprises dans l'intervalle] -1,1[. Le but de ce calcul est de proposer des valeurs à intégrer dans la mémoire.

Le deuxième calcul nous permet de trouver les valeurs de i_t. La fonction d'activation σ est la fonction sigmoïde qui renvoie des valeurs entre 0 et 1. Le but de ce calcul est de filtrer la quantité d'informations que l'on va envoyer dans le mémoire. C'est pour cela que l'on multiplie les 2 résultats obtenus. Nous allons donc obtenir un vecteur qui va ensuite être ajouté à la mémoire existante.

Pour récapituler, à ce moment précis, la mémoire que l'on note $C_t \; est \; égale$ à $C_{t-1}f_t + i_t\widetilde{C_t}$ avec C_{t-1} la mémoire au temps t-1, f_t qui correspond au résultat du forget gate donc de l'oubli d'information. A ceci s'ajoutent de nouvelles informations $i_t\widetilde{C_t}$.

3.3. Output Gate

Le but de l'output gate est de définir l'état de la cellule au temps t que l'on note h_t .

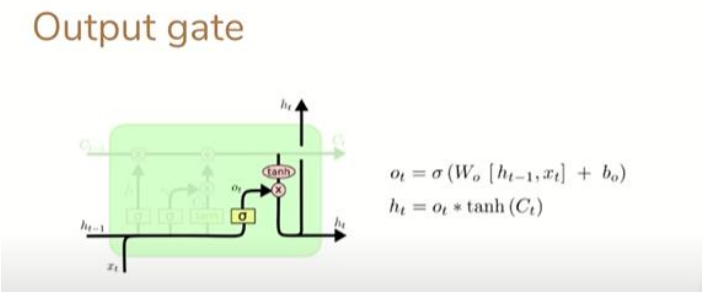

Figure 18 Zoom sur l'output gate[40]

L'output gate est divisé en 2 calculs. C_t, la mémoire de la cellule, va passer par une fonction d'activation tanh qui va renvoyer des valeurs entre -1 et 1. Ces valeurs seront ensuite multipliées par le vecteur o_t qui va sélectionner les informations de la mémoire utile.

L'état de la cellule dépend de la mémoire de cette cellule. Seules les informations utiles de la mémoire vont être utilisées, par exemple, si nous prenons une phrase et que nous voulons que le RNA nous donne le prochain mot. Toutes les informations utiles de la mémoire ne le seront pas forcément pour définir le mot suivant. L'état de la cellule va alors piocher dans la mémoire les mots qui lui seront utiles pour renvoyer la suite en sortie.

[40] Thibault Neveu, *Comprendre les LSTM - Réseaux de neurones récurrents.*

3.4. Les réseaux de neurones LSTM pour la classification

Nous venons de voir que les réseaux de neurones LSTM peuvent prédire une séquence au temps t+1 en gardant en mémoire des informations précédentes. Cependant, lorsque l'on fait de la classification, nous voulons obtenir comme résultat une classe et non une séquence.

Prenons l'exemple d'une suite de mots à prédire par le réseau de neurones LSTM et que nous obtenons comme phrase finale « Ce film était très beau », nous n'avons pas fait un travail de classification. Ce travail consisterait à dire si la phrase a une connotation positive, neutre ou négative par exemple.

Dans un travail de classification, après l'extraction d'informations sur les dépendances à long et à court terme dans les séries temporelles à l'aide des couches LSTM, ces informations se trouvent dans le dernier hidden layer de la dernière couche LSTM. Elles vont ensuite se propager dans des couches DENSE dont la dernière couche s'achève avec une fonction d'activation sigmoïde dans le cas d'une classification binaire ou softmax dans le cas d'une classification multi-classes. Ces fonctions ont pour but de transformer les valeurs prédites de la dernière couche à des valeurs comprises entre 0 et 1 (probabilités).

PARTIE III
INTERPRETATION DES RESULTATS

1. Présentation du jeu de données

Les données recueillies pour cette étude proviennent du jeu de données WISDM_ar_v1.1[41]. Ces données ont été présentées dans un article de Jennifer R. Kwapisz, Gary M. Weiss, Samuel A. Moore [42]. Ces données ont été récoltées sur 29 participants. Ils ont mis un téléphone muni d'un accéléromètre dans leur poche de pantalon de devant et ont fait différentes activités (marcher, courir, monter les escaliers, descendre les escaliers, s'assoir, se lever). Les données ont été récoltées toutes les 50 ms et à une fréquence de 20 hz. Les axes x, y, z de l'accéléromètre ainsi que les activités correspondantes ont été répertoriés à l'aide d'une application (Annexe 1).

Les données obtenues sont des données temporelles (Annexe 2). Elles sont au nombre de 1 098 203. Certaines activités sont beaucoup plus représentées que d'autres (Annexe 3 et 4). En effet, nous remarquons que l'activité « marcher » a été recensée plus de 400 000 fois alors que l'activité « s'assoir » l'a été moins de 50 000 fois. Notre modèle pourrait donc être biaisé, c'est-à-dire qu'il lui serait plus facile de prédire la marche que le fait de s'assoir. Or, le but de ce modèle est de prédire le plus possible chaque activité. Nous avons donc supprimé les données supplémentaires afin d'obtenir le même nombre de données pour chaque activité, à savoir 48 395. Notre base de données compte donc un total de 290 370 observations.

[41] « WISDM Lab: Wireless Sensor Data Mining ».
[42] Kwapisz et al.- 2011- Activity recognition using cell phone acceleromete

2. Création d'un modèle de réseau de neurones profonds pour la classification

Nous avons créé un modèle de réseau de neurones profonds. Plus le nombre d'époques augmente et plus l'erreur quadratique baisse (Annexe 5). Cela signifie que notre modèle est de plus en plus efficace pour prédire les données d'entrainement. Il est également efficace pour les données de test (Annexe 6). Notre modèle arrive à prédire correctement 87,67 % des activités sur des données qu'il n'a jamais vues et a une erreur quadratique moyenne de 1,14. Il a tout de même des difficultés à prédire certaines activités (Annexe 7). Il n'arrive à prédire l'activité « monter les escaliers » qu'à 66 % et « descendre les escaliers » qu'à 73 %. Il confond ces deux activités ce qui crée cette mauvaise prédiction.

3. Création d'un modèle de réseau de neurones convolutifs pour la classification

Nous avons créé un modèle de réseau de neurones convolutifs. Comme précédemment, notre réseau est efficace pour prédire les données d'entrainement (Annexe 8). L'erreur quadratique baisse lorsque le nombre d'époques augmente. Ce modèle est également efficace pour prédire des données encore jamais vues (Annexe 9). Il arrive à prédire correctement 90 % et a une erreur quadratique moyenne de 0,59. Il a quand même du mal à prédire certaines activités (Annexe 10). En effet, l'activité « descendre les escaliers » n'est prédite correctement qu'à 74 %. Comme précédemment, notre modèle confond monter et descendre les escaliers.

4. Création d'un modèle de réseau de neurones LSTM pour la classification.

Nous avons créé un modèle de réseau de neurones LSTM pour la classification de séries temporelles comme expliqué dans la partie 2, titre 3.4. Ce modèle prédit correctement les données d'entrainement (Annexe 11). L'erreur quadratique est de plus en plus faible lorsque le nombre d'époques augmente. Notre réseau est également très efficace pour la prédiction de données de test (Annexe 12). Il prédit correctement 95,25 % des activités et a une erreur quadratique moyenne de 0,1772. Il confond comme les modèles précédents les activités monter et descendre les escaliers même si celles-ci sont prédites correctement à environ 90 % (Annexe 13).

5. Discussion

Nos données ont été recueillies sur des intervalles de 50 ms. Cela signifie qu'il y a un lien entre toutes les données récoltées. La différence au niveau de la prédiction et de l'erreur quadratique de nos différents modèles peut s'expliquer par ce lien. En effet, le modèle de réseau de neurones profond ne va pas remarquer de lien entre toutes les données et ne va donc pas les traiter comme des données temporelles. Au contraire, le réseau de neurones LSTM va reconnaitre ces données temporelles pour obtenir une prédiction plus fiable des activités humaines.

CONCLUSION

Ce mémoire avait pour but de classifier des séries temporelles par réseaux de neurones LSTM. Il avait également pour but de comparer les résultats obtenus avec ceux d'autres réseaux.

Nous avons vu la distinction entre réseaux feed-forward et récurrents. Les réseaux de neurones LSTM sont des réseaux récurrents qui ont la particularité d'avoir une mémoire à long terme ce qui pallie le problème de rétropropagation du gradient des autres réseaux récurrents.

Les réseaux de neurones LSTM sont utilisés pour prédire une séquence dans le temps. Pour faire de la classification avec ces réseaux, il faut rajouter aux couches LSTM des couches denses qui permettront ce travail de classification.

Nous avons créé 3 modèles pour classifier l'activité humaine, un réseau profond, un réseau convolutif et un réseau LSTM. Nous obtenons une prédiction correcte des activités de 95 % pour le réseau LSTM, de 90 % pour le réseau convolutif et de 87 % pour le réseau profond.

Les réseaux de neurones LSTM sont donc les plus adaptés pour classifier des séries temporelles.

BIBLIOGRAPHIE

[1] ALLIANCE, l'European Dana. La cellule nerveuse (neurone). Dans : *Futura* [en ligne]. [s. d.]. [Consulté le 20 mars 2021]. Disponible à l'adresse : https://www.futura-sciences.com/sante/dossiers/medecine-voyage-cerveau-525/page/3/

[2] Le neurone. Dans : *Fédération pour la Recherche sur le Cerveau (FRC)* [en ligne]. [s. d.]. [Consulté le 20 mars 2021]. Disponible à l'adresse : https://www.frcneurodon.org/comprendre-le-cerveau/a-la-decouverte-du-cerveau/le-neurone/

[3] [5] *Dendrites - un aperçu | Sujets ScienceDirect* [en ligne]. [s. d.]. [Consulté le 20 mars 2021]. Disponible à l'adresse : https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/dendrites

[4] *Structure - L'axone* [en ligne]. [s. d.]. [Consulté le 20 mars 2021]. Disponible à l'adresse : http://lyrobossite.free.fr/Structure_II_L'axone.htm

[6] *L'explication la plus facile - Neurones et Transmission Neuronale* [en ligne]. 2019. [Consulté le 20 mars 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=iVcJD1N0bRA

[7] *Corpus : Au cœur des organes. Le système nerveux* [en ligne]. 2018. [Consulté le 20 mars 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=oK3esXMQxaI

[8] *Le Neurone- cellule fondamentale du syst*  *me nerveux.* [en ligne]. [s. d.]. [Consulté le 20 mars 2021]. Disponible à l'adresse : http://www.corpshumain.ca/Neurone.php

[9] *La communication entre les neurones : la transmission synaptique - SVT - 4e* [en ligne]. [s. d.]. [Consulté le 20 mars 2021]. Disponible à l'adresse : https://www.assistancescolaire.com/eleve/4e/svt/reviser-une-notion/la-communication-entre-les-neurones-la-transmission-synaptique-3sad06

[10] *D4. Mécanisme neuronal de transmission de l'information - ppt télécharger* [en ligne]. [s. d.]. [Consulté le 28 mars 2021]. Disponible à l'adresse : https://slideplayer.fr/slide/10798158/

[11] *LE CERVEAU À TOUS LES NIVEAUX!* [en ligne]. [s. d.]. [Consulté le 21 mars 2021]. Disponible à l'adresse : https://lecerveau.mcgill.ca/flash/a/a_01/a_01_cl/a_01_cl_ana/a_01_cl_ana.html

[12] *Les cellules nerveuses et la moelle épinière* [en ligne]. [s. d.]. Disponible à l'adresse : https://edu.ge.ch/decandolle/sites/localhost.decandolle/files/diaporama-sn2-cellnervetme_0.pdf

[13] *Sciences et technologies* [en ligne]. [s. d.]. [Consulté le 21 mars 2021]. Disponible à l'adresse : https://scientificsentence.net/Equations/Sciences_Technologies/index.php?key=yes&Integer=systeme_nerveux_central

[14] USITO. Usito. Dans : *Usito* [en ligne]. [s. d.]. [Consulté le 21 mars 2021]. Disponible à l'adresse : https://usito.usherbrooke.ca/définitions/interneurone

[15] *Neurone pseudo-unipolaire* [en ligne]. [S. l.] : [s. n.], 14 octobre 2020. [Consulté le 21 mars 2021]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Neurone_pseudo-unipolaire&oldid=175558052. Page Version ID: 175558052

[16] Système nerveux central | Système nerveux, Système nerveux central, Physiologie. Dans : *Pinterest* [en ligne]. [s. d.]. [Consulté le 21 mars 2021]. Disponible à l'adresse : https://www.pinterest.com/pin/670614200740898644/

[17] Polarisation des neurones. Dans : *Celluloyd* [en ligne]. [s. d.]. [Consulté le 21 mars 2021]. Disponible à l'adresse : https://celluloyd.tumblr.com/post/171570066570/polarisation-des-neurones-les-neurones-sont-les

[18] [20] L, +Bastien. Réseau de neurones artificiels : qu'est-ce que c'est et à quoi ça sert ? Dans : *LeBigData.fr* [en ligne]. 5 avril 2019. [Consulté le 26 mars 2021]. Disponible à l'adresse : https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition

[19] SCHMITT, A., LE BLANC, B., CORSINI, M.-M., LAFOND, C. et BRUZEK, J. Les réseaux de neurones artificiels. Un outil de traitement de données prometteur pour l'anthropologie. *Bulletins et mémoires de la Société d'Anthropologie de Paris* [en ligne]. Société d'anthropologie de Paris, Juin 2001, n° 13 (1-2). [Consulté le 26 mars 2021]. Disponible à l'adresse : http://journals.openedition.org/bmsap/4463

[21] SCIENCE4ALL. *Les réseaux de neurones | Intelligence artificielle 41* [en ligne]. 17 septembre 2018. [Consulté le 26 mars 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=8qL2lSQd9L8&t=94s

[22] CHRONOPHAGE. *Les maths des réseaux de neurones - Introduction au Deep Learning #2* [en ligne]. 2019. [Consulté le 27 mars 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=QuMabWInlAQ

[23] LEMAIRE, Vincent, BERNIER, Olivier, COLLOBERT, Daniel et CLÉROT, Fabrice. *Une nouvelle fonction de coût régularisante dans les réseaux de neurones artificiels*. [s. d.], p. 25

[24] *Rétropropagation de gradient, analyse. Le deep learning* [en ligne]. [s. d.]. [Consulté le 13 mai 2021]. Disponible à l'adresse : https://www.nagwa.com/fr/videos/724185368324/

[25] Deep learning : la rétropropagation du gradient. Dans : *Miximum* [en ligne]. 22 février 2017. [Consulté le 17 mai 2021]. Disponible à l'adresse : https://www.miximum.fr/blog/introduction-au-deep-learning-2/

[26] Démystifier le Machine Learning, Partie 2 : les Réseaux de Neurones artificiels. Dans : *Juri'Predis* [en ligne]. [s. d.]. [Consulté le 25 mars 2021]. Disponible à l'adresse : https://www.juripredis.com/fr/blog/id-19-demystifier-le-machine-learning-partie-2-les-reseaux-de-neurones-artificiels

[27] *Réseau de neurones : perceptron monocouche - IA - IAD - Java : Supports de cours* [en ligne]. [s. d.]. [Consulté le 27 mars 2021]. Disponible à l'adresse : http://emmanuel.adam.free.fr/site/spip.php?article183

[28] RAKOTOMALALA, Ricco. *Perceptrons simples et multicouches*. [s. d.]. Disponible à l'adresse : Rakotomalala - Perceptrons simples et multicouches.pdf

[29] Réseau de neurones convolutifs. Dans : *Data Analytics Post* [en ligne]. [s. d.]. [Consulté le 27 mars 2021]. Disponible à l'adresse : https://dataanalyticspost.com/Lexique/reseau-de-neurones-convolutifs/

[30] Les réseaux de neurones convolutifs. Dans : *Natural Solutions : Gestion de données de biodiversité* [en ligne]. [s. d.]. [Consulté le 14 mai 2021]. Disponible à l'adresse : https://www.natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-neurones-convolutifs

[31] Convolutional neural network | Deep Learning | DataScientest. Dans : *Formation Data Science | DataScientest.com* [en ligne]. 25 juin 2020. [Consulté le 14 mai 2021]. Disponible à l'adresse : https://datascientest.com/convolutional-neural-network

[32] Réseaux de neurones récurrents. Dans : *Data Analytics Post* [en ligne]. [s. d.]. [Consulté le 27 mars 2021]. Disponible à l'adresse : https://dataanalyticspost.com/Lexique/reseaux-de-neurones-recurrents/

[33] [41] *WISDM Lab: Wireless Sensor Data Mining* [en ligne]. [s. d.]. [Consulté le 15 mai 2021]. Disponible à l'adresse : https://www.cis.fordham.edu/wisdm/

[34] USER, Super. *Réseaux neuronaux récurrents et LSTM* [en ligne]. [s. d.]. [Consulté le 28 mars 2021]. Disponible à l'adresse : https://datasciencetoday.net/index.php/fr/machine-learning/148-reseaux-neuronaux-recurrents-et-lstm

[35] THIBAULT NEVEU. *Comprendre les réseaux de neurones récurrents (RNN)* [en ligne]. 11 janvier 2019. [Consulté le 27 mars 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=EL439RMv3Xc

[36] HOCHREITER, Sepp et SCHMIDHUBER, Jürgen. Long Short-Term Memory. *Neural Computation* [en ligne]. Novembre 1997, Vol. 9, n° 8, p. 1735-1780. [Consulté le 28 mars 2021]. DOI 10.1162/neco.1997.9.8.1735

[37] *Réseau de neurones récurrents* [en ligne]. [S. l.] : [s. n.], 11 mars 2020. [Consulté le 28 mars 2021]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=R%C3%A9seau_de_neurones_r%C3%A9currents&oldid=168319282. Page Version ID: 168319282

[38] [39] [40] THIBAULT NEVEU. *Comprendre les LSTM - Réseaux de neurones récurrents* [en ligne]. 2019. [Consulté le 28 mars 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=3xgYxrNyE54

[42] KWAPISZ, Jennifer R., WEISS, Gary M. et MOORE, Samuel A. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter* [en ligne]. Mars 2011, Vol. 12, n° 2, p. 74-82. [Consulté le 15 mai 2021]. DOI 10.1145/1964897.1964918

ANNEXES

TABLE DES ANNEXES

Annexe 1. Présentation de la base de données

	activity	time	x	y	z
0	Jogging	49105962326000	-0.694638	12.680544	0.503953
1	Jogging	49106062271000	5.012288	11.264028	0.953424
2	Jogging	49106112167000	4.903325	10.882658	-0.081722
3	Jogging	49106222305000	-0.612916	18.496431	3.023717
4	Jogging	49106332290000	-1.184970	12.108489	7.205164

Table 1 : Base de données. **Cohorte** : 29 participants. **Lecture** : La première donnée enregistrée a pour axe x -0.6, pour axe y 12.6 et pour axe z 0.5. Ces données correspondent à l'activité jogging.

Dans cette étude, 29 participants ont porté un téléphone portable au niveau de la taille et on fait différentes activités (marcher, courir, monter les escaliers, descendre les escaliers, s'assoir, se lever). Les données (axe x, y, z de l'accéléromètre du téléphone) ont été recueillies toutes les 50 ms à une fréquence de 20 hz et l'activité correspondante a été répertoriée. Ce jeu de données comprend 1 098 203 observations.

Annexe 2. Représentation des signaux de l'activité « marcher » pour un intervalle de 10 secondes

Table 2 : Représentation des signaux de l'activité « marcher » pour un intervalle de 10 secondes. **Cohorte** : 29 participants. **Lecture** : Les signaux de l'axe x varient entre -5 et 5 (10 = 1g = 9.81 m/s^2).

Nous remarquons que toutes les données de l'accéléromètre (x, y, z) varient au cours du temps. Nous aurions pu penser que seul l'axe des x serait impacté par la marche.

Annexe 3. Répartition graphique des différentes activités

Table 3 : Répartition graphique des différentes activités. **Cohorte** : 29 participants **Lecture** : L'activité walking a été recensée plus de 400 000 fois.

Nous remarquons que le jeu de données est déséquilibré puisque nous avons répertorié beaucoup trop d'activités de marche et de course comparé à l'activité de s'assoir et de se lever.

Annexe 4. Répartition des différentes activités

```
Walking       424397
Jogging       342176
Upstairs      122869
Downstairs    100427
Sitting        59939
Standing       48395
Name: activity, dtype: int64
```

Table 4 : Répartition des différentes activités. **Cohorte** : 29 participants. **Lecture** : Nous recensons l'activité standing 48 395 fois.

Nous avons 10 fois plus de données de marche que de personnes qui se lèvent. Nous allons supprimer les données supplémentaires pour que notre modèle ne soit pas biaisé. Nous allons donc garder 48 395 observations par activités. Notre jeu de données comptera désormais 290 370 observations.

Annexe 5. Graphique de l'erreur quadratique en fonction du nombre d'époques pour le réseau de neurones profonds

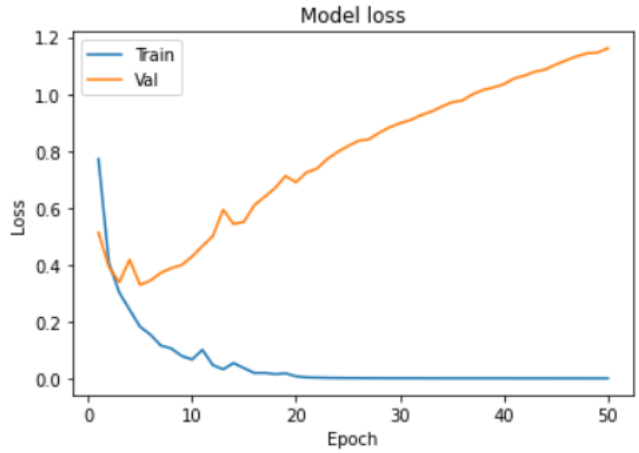

Table 5 : Graphique de l'erreur quadratique en fonction du nombre d'époques pour le réseau de neurones profonds. **Cohorte** : 29 participants. **Lecture** : L'erreur quadratique des données d'entrainement est environ égale à 0 lorsque le nombre d'époques devient supérieur à 20.

Nous remarquons que plus les époques augmentent et plus l'erreur quadratique des données d'entrainement est de plus en plus faible. Ceci est logique car notre modèle a déjà vu de nombreuses fois ces données. Au contraire, pour les données de test, on remarque que l'erreur quadratique est beaucoup plus importante.

Annexe 6. Evaluation des données de test pour le réseau de neurones profonds

```
model_m.evaluate(X_test, y_test)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 240) for input KerasTensor(t
46/46 [==============================] - 0s 6ms/step - loss: 1.1439 - accuracy: 0.8767
[1.143941879272461, 0.8767217397689819]
```

Table 6 : Evaluation des données de test pour le réseau de neurones profonds. **Cohorte** : 29 participants. **Lecture** : L'erreur quadratique moyenne des données de test est de 1.14.

Nous remarquons que ce modèle de réseau de neurones profond prédit correctement l'activité à 87,67% et possède une erreur quadratique de 1,14 (pour les données de test). Nous pouvons donc dire que ce réseau de neurones fonctionne bien pour la classification.

Annexe 7. Matrice de confusion normalisée pour la prédiction d'activités avec le réseau de neurones profonds

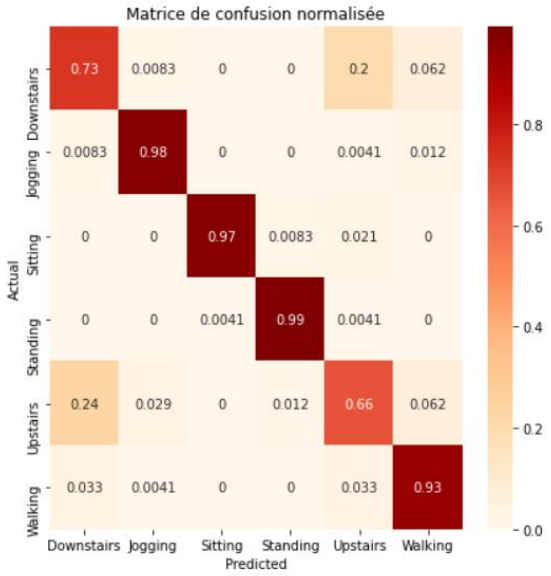

Table 7 : Matrice de confusion normalisée pour la prédiction d'activités avec le réseau de neurones profonds. **Cohorte** : 29 participants. **Lecture** : L'activité downstairs est prédite correctement à 73 %.

Nous remarquons que ce modèle prédit très bien la course, la marche, le fait de s'assoir et de se lever (à plus de 90 %). Pourtant, ce modèle confond le fait de monter les escaliers et de les descendre.

Annexe 8. Graphique de l'erreur quadratique en fonction du nombre d'époques pour le réseau de neurones convolutifs

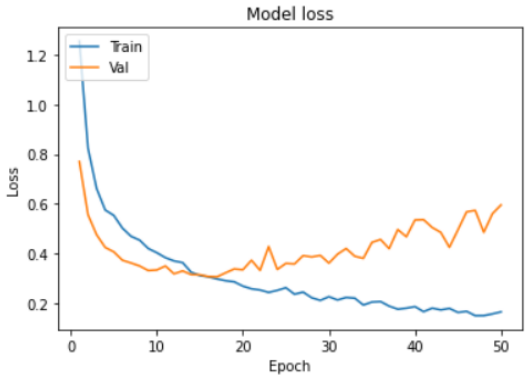

Table 8 : Graphique de l'erreur quadratique en fonction du nombre d'époques pour le réseau de neurones convolutifs. **Cohorte** : 29 participants. **Lecture** : L'erreur quadratique des données d'entrainement est inférieure à 0.2 à partir de 30 époques.

Nous remarquons que l'erreur quadratique des données d'entrainement est plus faible que celle des données de test. Nous remarquons également que l'erreur quadratique des données de test est comprise entre 0,35 et 0,8 ce qui est bien meilleur qu'avec le réseau de neurones profonds vu précédemment.

Annexe 9. Evaluation des données de test pour le réseau de neurones convolutifs

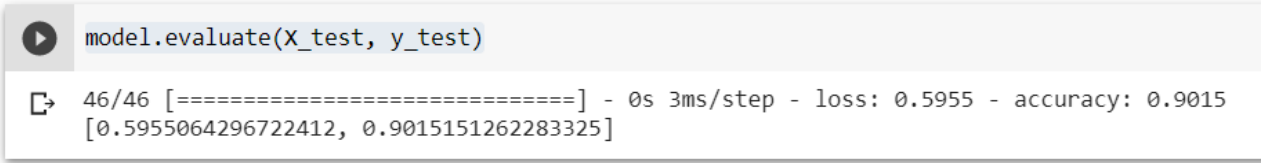

Table 9 : Evaluation des données de test pour le réseau de neurones convolutifs. **Cohorte** : 29 participants. **Lecture** : L'erreur quadratique moyenne est de 0 .59.

Nous remarquons que ce modèle de réseau de neurones convolutifs prédit correctement l'activité à 90,15 % et possède une erreur quadratique de 0.5955 (pour les données de test). Nous pouvons donc dire que ce réseau de neurone fonctionne mieux que le précédent que ce soit au niveau de l'erreur quadratique ou de la précision.

Annexe 10. Matrice de confusion normalisée pour la prédiction d'activités avec le réseau de neurones convolutifs

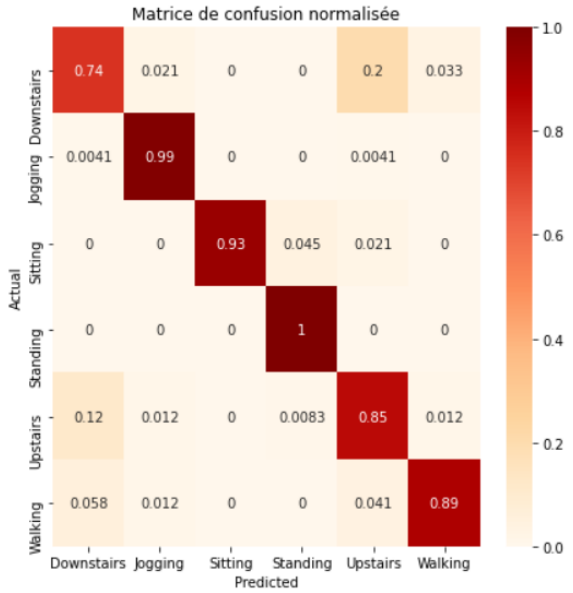

Table 10 : Matrice de confusion normalisée pour la prédiction d'activités avec le réseau de neurones convolutifs. **Cohorte** : 29 participants. **Lecture** : L'activité downstairs est prédite correctement à 74 %.

Nous remarquons que les activités course, s'assoir et se lever sont très bien prédites. Nous remarquons également que le réseau confond les 3 activités marche, monter et descendre les escaliers. Malgré cette confusion, les prédictions sont tout de même meilleures qu'avec un réseau de neurones profonds.

Annexe 11. Graphique de l'erreur quadratique en fonction du nombre d'époques pour le réseau de neurones LSTM

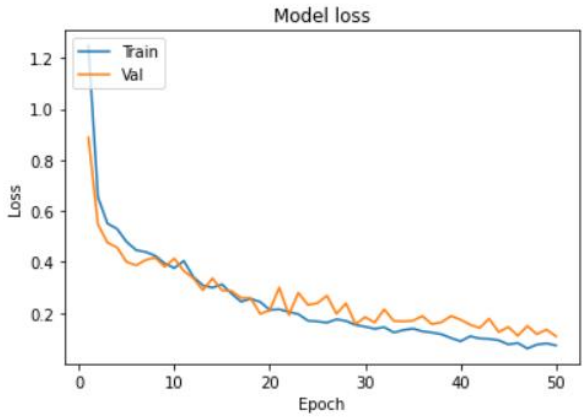

Table 11 : Graphique de l'erreur quadratique en fonction du nombre d'époques pour le réseau de neurones LSTM. **Cohorte** : 29 participants. **Lecture** : L'erreur quadratique des données d'entrainement est environ égale à 0.2 pour 20 époques.

Nous remarquons que la courbe de l'erreur quadratique des données de validation en fonction du nombre d'époques semble suivre celle des données d'entrainement. Cela montre la fiabilité du modèle pour des données qu'il n'a pas encore vues.

Annexe 12. Evaluation des données de test pour le réseau de neurones LSTM

```
model.evaluate(X_test, y_test)
```

```
46/46 [==============================] - 1s 21ms/step - loss: 0.1772 - acc: 0.9525
[0.17721110582351685, 0.952479362487793]
```

Table 12 : Evaluation des données de test pour le réseau de neurones LSTM. **Cohorte** : 29 participants. **Lecture** : L'erreur quadratique moyenne est de 0.17

Nous remarquons que ce modèle de réseau de neurones LSTM prédit correctement l'activité à 95,25% et possède une erreur quadratique de 0.1772 (pour les données de test). Nous pouvons donc dire que ce réseau de neurones fonctionne mieux que les précédents que ce soit au niveau de l'erreur quadratique ou de la précision.

Annexe 13. Matrice de confusion normalisée pour la prédiction d'activités avec le réseau de neurones LSTM

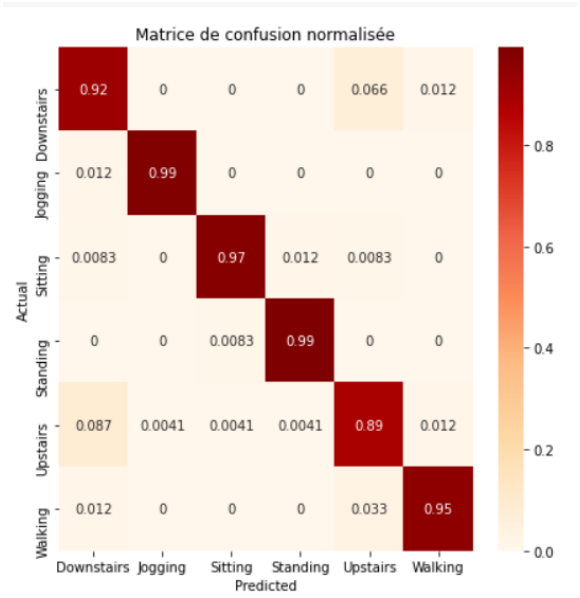

Table 13 : Matrice de confusion normalisée pour la prédiction d'activités avec le réseau de neurones LSTM. **Cohorte** : 29 participants. **Lecture** : L'activité downstairs est prédite correctement à 92 %.

Nous remarquons que le réseau de neurones confond toujours un peu le fait de monter et de descendre les escaliers. Il reste quand même très fiable puisque la précision de la prédiction de chaque activité est supérieure ou égale à 89 %.

Annexe 14. Code pour le réseau de neurones profonds.

```
!pip install detecta
from __future__ import print_function
from matplotlib import pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
from IPython.display import display, HTML
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn import preprocessing
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.optimizers import Adam
from sklearn import preprocessing
from tensorflow.keras import layers
import tensorflow as tf
from tensorflow.keras.layers import Dense, LSTM, InputLayer, Dropout, BatchNormalization
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import to_categorical
from matplotlib import pyplot

column_names = [
  'user',
  'activity',
```

```
  'time',
  'x',
  'y',
  'z'
]
df = pd.read_csv(
  'WISDM_ar_v1.1_raw.txt',
  header=None,
  names=column_names
)
df.z.replace(regex=True, inplace=True, to_replace=r';', value=r'')
df['z'] = df.z.astype(np.float64)
df.dropna(axis=0, how='any', inplace=True)
df.shape

df.head
Fs=20 # fréquence des signaux de 20 hz
activities = df['activity'].value_counts().index
activities
df = df.drop(['user'], axis = 1).copy()
df.head()
df = df.drop(['time'], axis = 1).copy()
df.head()
df['activity'].value_counts().plot(kind='bar',
                                   title='Training Examples by Activity Type')
plt.show()
df['activity'].value_counts()
Walking = df[df['activity']=='Walking'].head(48395).copy()
Jogging = df[df['activity']=='Jogging'].head(48395).copy()
Upstairs = df[df['activity']=='Upstairs'].head(48395).copy()
Downstairs = df[df['activity']=='Downstairs'].head(48395).copy()
Sitting = df[df['activity']=='Sitting'].head(48395).copy()
Standing = df[df['activity']=='Standing'].copy()

balanced_data = pd.DataFrame()
balanced_data = balanced_data.append([Walking, Jogging, Upstairs, Downstairs, Sitting, Standing])
balanced_data.shape
balanced_data['activity'].value_counts()
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
balanced_data['label'] = label.fit_transform(balanced_data['activity'])
balanced_data.head()
label.classes_ # nom des colonnes
X = balanced_data[['x', 'y', 'z']]
```

```
y = balanced_data['label']
X.shape
y.shape
scaler = StandardScaler()
X = scaler.fit_transform(X)

scaled_X = pd.DataFrame(data = X, columns = ['x', 'y', 'z'])
scaled_X['label'] = y.values

scaled_X
import scipy.stats as stats
Fs = 20  # 20 hz
frame_size = Fs*4 # durée =4 sec d'où 20*4
hop_size = Fs*2 # 40

def get_frames(df, frame_size, hop_size):

    N_FEATURES = 3

    frames = []
    labels = []
    for i in range(0, len(df) - frame_size, hop_size):
        x = df['x'].values[i: i + frame_size]
        y = df['y'].values[i: i + frame_size]
        z = df['z'].values[i: i + frame_size]

        # Retrieve the most often used label in this segment
        label = stats.mode(df['label'][i: i + frame_size])[0][0]
        frames.append([x, y, z])
        labels.append(label)

    # Bring the segments into a better shape
    frames = np.asarray(frames).reshape(-1, frame_size, N_FEATURES)
    labels = np.asarray(labels)

    return frames, labels
X, y = get_frames(scaled_X, frame_size, hop_size)

X.shape, y.shape
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)
X_train.shape, X_test.shape,y_train.shape,y_test.shape
y_train = y_train.reshape(5806, 1)
y_test = y_test.reshape(1452, 1)
from sklearn.preprocessing import OneHotEncoder
```

```
enc = OneHotEncoder(handle_unknown='ignore', sparse=False)
enc = enc.fit(y_train)
y_train = enc.transform(y_train)
y_test = enc.transform(y_test)
y_train.shape,y_test.shape
num_time_periods, num_sensors = X_train.shape[1], X_train.shape[2]
input_shape = (num_time_periods*num_sensors)
X_train = X_train.reshape(X_train.shape[0], input_shape)
X_train.shape
X_train = X_train.astype('float32')
y_train = y_train.astype('float32')
X_train.shape,y_train.shape
TIME_PERIODS = 80

model_m = Sequential()
model_m.add(Reshape((TIME_PERIODS, 3), input_shape=(input_shape,)))
model_m.add(Dense(100, activation='relu'))
model_m.add(Dense(100, activation='relu'))
model_m.add(Dense(100, activation='relu'))
model_m.add(Flatten())
model_m.add(Dense(6, activation='softmax'))

BATCH_SIZE = 32
EPOCHS = 50


model_m.compile(loss='categorical_crossentropy',
            optimizer='adam', metrics=['accuracy'])

history = model_m.fit(X_train,
                  y_train,
                  batch_size=BATCH_SIZE,
                  epochs=EPOCHS,
                  validation_split=0.2,
                  verbose=1)
def plot_learningCurve(history, epochs):
  epoch_range = range(1, epochs+1)
  # Plot training & validation loss values
  plt.plot(epoch_range, history.history['loss'])
  plt.plot(epoch_range, history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
```

```
  plt.show()
plot_learningCurve(history, 50)
pip install mlxtend
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
model_m.evaluate(X_test, y_test)
y_pred = model_m.predict(X_test)
y_pred.shape,y_test.shape,X_test.shape
from sklearn.metrics import confusion_matrix

def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  cm=cm/ cm.astype(np.float).sum(axis=1)
  fig, ax = plt.subplots(figsize=(7, 7))
  ax = sns.heatmap(
     cm,
     annot=True,
     ax=ax,
     cmap="OrRd"
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.title('Matrice de confusion normalisée')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  plt.show()
plot_cm(
  enc.inverse_transform(y_test),
  enc.inverse_transform(y_pred),label.classes_

)
def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  fig, ax = plt.subplots(figsize=(7, 7))
  ax = sns.heatmap(
```

```
    cm,
    annot=True,
    ax=ax,
    cmap="OrRd",
    fmt='g'
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.title('Matrice de confusion non normalisée')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  plt.show()
plot_cm(
  enc.inverse_transform(y_test),
  enc.inverse_transform(y_pred),label.classes_

)
```

Ce code est inspiré du document Time Series Classification for Human Activity Recognition with LSTMs in Keras. Dans : *Curiousily* [en ligne]. [s. d.]. [Consulté le 15 avril 2021]. Disponible à l'adresse : https://curiousily.com/posts/time-series-classification-for-human-activity-recognition-with-lstms-in-keras/

Annexe 15. Code pour le réseau de neurones convolutifs

```
!pip install detecta
from __future__ import print_function
from matplotlib import pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
from IPython.display import display, HTML

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn import preprocessing

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.optimizers import Adam
from sklearn import preprocessing
column_names = [
  'user',
  'activity',
  'time',
  'x',
  'y',
  'z'
]
df = pd.read_csv(
  'WISDM_ar_v1.1_raw.txt',
  header=None,
  names=column_names
)
df.z.replace(regex=True, inplace=True, to_replace=r';', value=r'')
df['z'] = df.z.astype(np.float64)
df.dropna(axis=0, how='any', inplace=True)
```

```
df.shape
df.shape
df.info()
df.isnull().sum()
Fs=20 # fréquence des signaux de 20 hz
activities = df['activity'].value_counts().index
activities
def plot_activity(activity, df):
    fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(15, 7), sharex=True)
    plot_axis(ax0, df['time'], df['x'], 'X-Axis')
    plot_axis(ax1, df['time'], df['y'], 'Y-Axis')
    plot_axis(ax2, df['time'], df['z'], 'Z-Axis')
    plt.subplots_adjust(hspace=0.2)
    fig.suptitle(activity)
    plt.subplots_adjust(top=0.90)
    plt.show()

def plot_axis(ax, x, y, title):
    ax.plot(x, y, 'g')
    ax.set_title(title)
    ax.xaxis.set_visible(False)
    ax.set_ylim([min(y) - np.std(y), max(y) + np.std(y)])
    ax.set_xlim([min(x), max(x)])
    ax.grid(True)

for activity in activities:
    df_for_plot = df[(df['activity'] == activity)][:Fs*10]
    plot_activity(activity, df_for_plot)
df = df.drop(['user', 'time'], axis = 1).copy()
df.head()
df['activity'].value_counts().plot(kind='bar',
                                   title='Training Examples by Activity Type')
plt.show()
df['activity'].value_counts()
Walking = df[df['activity']=='Walking'].head(48395).copy()
Jogging = df[df['activity']=='Jogging'].head(48395).copy()
Upstairs = df[df['activity']=='Upstairs'].head(48395).copy()
Downstairs = df[df['activity']=='Downstairs'].head(48395).copy()
Sitting = df[df['activity']=='Sitting'].head(48395).copy()
Standing = df[df['activity']=='Standing'].copy()

balanced_data = pd.DataFrame()
balanced_data = balanced_data.append([Walking, Jogging, Upstairs, Downstairs, Sitting, Standing])
balanced_data.shape
```

```
balanced_data['activity'].value_counts()
balanced_data.head()
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
balanced_data['label'] = label.fit_transform(balanced_data['activity'])
balanced_data.head()
label.classes_ # nom des colonnes
X = balanced_data[['x', 'y', 'z']]
y = balanced_data['label']
X.shape
y.shape
scaler = StandardScaler()
X = scaler.fit_transform(X)

scaled_X = pd.DataFrame(data = X, columns = ['x', 'y', 'z'])
scaled_X['label'] = y.values

scaled_X
df.info()
import scipy.stats as stats
Fs = 20  # 20 hz
frame_size = Fs*4 # durée =4 sec d'où 20*4
hop_size = Fs*2 # 40
def get_frames(df, frame_size, hop_size):

    N_FEATURES = 3

    frames = []
    labels = []
    for i in range(0, len(df) - frame_size, hop_size):
        x = df['x'].values[i: i + frame_size]
        y = df['y'].values[i: i + frame_size]
        z = df['z'].values[i: i + frame_size]

        # Retrieve the most often used label in this segment
        label = stats.mode(df['label'][i: i + frame_size])[0][0]
        frames.append([x, y, z])
        labels.append(label)

    # Bring the segments into a better shape
    frames = np.asarray(frames).reshape(-1, frame_size, N_FEATURES)
    labels = np.asarray(labels)

    return frames, labels
```

```
X, y = get_frames(scaled_X, frame_size, hop_size)

X.shape, y.shape
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)
X_train.shape, X_test.shape
X_train[0].shape, X_test[0].shape
X_train = X_train.reshape(5806, 80, 3, 1)
X_test = X_test.reshape(1452, 80, 3, 1)
X_train[0].shape, X_test[0].shape
pip install --upgrade tensorflow
import tensorflow as tf
model = Sequential()
model.add(Conv2D(64, (2, 2), activation = 'relu', input_shape = X_train[0].shape))
model.add(Dropout(0.1))


model.add(Flatten())
model.add(Dense(32, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(6, activation='softmax'))
model.compile(optimizer=Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy',
metrics = ['accuracy'])
history = model.fit(X_train, y_train, epochs = 50, validation_data= (X_test, y_test), verbose=1)
def plot_learningCurve(history, epochs):
  epoch_range = range(1, epochs+1)
  # Plot training & validation loss values
  plt.plot(epoch_range, history.history['loss'])
  plt.plot(epoch_range, history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()
plot_learningCurve(history, 50)
pip install mlxtend
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import plot_confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
model.evaluate(X_test, y_test)
y_pred = model.predict_classes(X_test)
y_test.shape,y_pred.shape,X_test.shape
from sklearn.metrics import confusion_matrix

def plot_cm(y_test, y_pred, class_names):
  cm = confusion_matrix(y_test, y_pred)
  cm=cm/ cm.astype(np.float).sum(axis=1)
  fig, ax = plt.subplots(figsize=(7, 7))
  ax = sns.heatmap(
     cm,
     annot=True,
     ax=ax,
     cmap="OrRd"
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.title('Matrice de confusion normalisée')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  plt.show()
plot_cm(
  y_test,
  y_pred,label.classes_

)
def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  fig, ax = plt.subplots(figsize=(7, 7))
  ax = sns.heatmap(
     cm,
     annot=True,
     ax=ax,
     cmap="OrRd",
     fmt='g'
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.title('Matrice de confusion non normalisée')
```

```
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  plt.show()
plot_cm(
  y_test,
  y_pred,label.classes_

)
```

Ce code est inspiré du document AARYA. Human Activity Recognition Using Accelerometer Data. Dans : *KGP Talkie* [en ligne]. 28 août 2020. [Consulté le 17 avril 2021]. Disponible à l'adresse : https://kgptalkie.com/human-activity-recognition-using-accelerometer-data/

Annexe 16. Code pour le réseau de neurones LSTM

```
!pip install detecta
from __future__ import print_function
from matplotlib import pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
from IPython.display import display, HTML

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn import preprocessing

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.optimizers import Adam
from sklearn import preprocessing
from tensorflow.keras import layers
import tensorflow as tf
from tensorflow.keras.layers import Dense, LSTM, InputLayer, Dropout, BatchNormalization
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import to_categorical
from matplotlib import pyplot
column_names = [
  'user',
```

```
 'activity',
 'time',
 'x',
 'y',
 'z'
]
df = pd.read_csv(
 'WISDM_ar_v1.1_raw.txt',
 header=None,
 names=column_names
)
df.z.replace(regex=True, inplace=True, to_replace=r';', value=r'')
df['z'] = df.z.astype(np.float64)
df.dropna(axis=0, how='any', inplace=True)
df.shape
df.head
df.info()
df.isnull().sum()
Fs=20 # fréquence des signaux de 20 hz
activities = df['activity'].value_counts().index
activities
df = df.drop(['user', 'time'], axis = 1).copy()
df.head()
df['activity'].value_counts().plot(kind='bar',
                                   title='Training Examples by Activity Type')
plt.show()
df['activity'].value_counts()
Walking = df[df['activity']=='Walking'].head(48395).copy()
Jogging = df[df['activity']=='Jogging'].head(48395).copy()
Upstairs = df[df['activity']=='Upstairs'].head(48395).copy()
Downstairs = df[df['activity']=='Downstairs'].head(48395).copy()
Sitting = df[df['activity']=='Sitting'].head(48395).copy()
Standing = df[df['activity']=='Standing'].copy()

balanced_data = pd.DataFrame()
balanced_data = balanced_data.append([Walking, Jogging, Upstairs, Downstairs, Sitting, Standing])
balanced_data.shape
balanced_data['activity'].value_counts()
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
balanced_data['label'] = label.fit_transform(balanced_data['activity'])
balanced_data.head()
label.classes_ # nom des colonnes
X = balanced_data[['x', 'y', 'z']]
```

```
y = balanced_data['label']
X.shape
y.shape
scaler = StandardScaler()
X = scaler.fit_transform(X)

scaled_X = pd.DataFrame(data = X, columns = ['x', 'y', 'z'])
scaled_X['label'] = y.values

scaled_X
import scipy.stats as stats
Fs = 20  # 20 hz
frame_size = Fs*4 # durée =4 sec d'où 20*4
hop_size = Fs*2 # 40
def get_frames(df, frame_size, hop_size):

    N_FEATURES = 3

    frames = []
    labels = []
    for i in range(0, len(df) - frame_size, hop_size):
        x = df['x'].values[i: i + frame_size]
        y = df['y'].values[i: i + frame_size]
        z = df['z'].values[i: i + frame_size]

        label = stats.mode(df['label'][i: i + frame_size])[0][0]
        frames.append([x, y, z])
        labels.append(label)
    frames = np.asarray(frames).reshape(-1, frame_size, N_FEATURES)
    labels = np.asarray(labels)

    return frames, labels
X, y = get_frames(scaled_X, frame_size, hop_size)

X.shape, y.shape
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)
X_train.shape, X_test.shape,y_train.shape,y_test.shape
X_train.shape[1]
X_train.shape[2]
y_train = y_train.reshape(5806, 1)
y_test = y_test.reshape(1452, 1)
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore', sparse=False)
enc = enc.fit(y_train)
```

```
y_train = enc.transform(y_train)
y_test = enc.transform(y_test)
y_train.shape,y_test.shape
model = keras.Sequential()
model.add(
   keras.layers.LSTM(
       units=120,
       input_shape=[X_train.shape[1], X_train.shape[2]]
    )
   )
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(units=120, activation='relu'))
model.add(keras.layers.Dense(units=120, activation='relu'))
model.add(keras.layers.Dense(y_train.shape[1], activation='softmax'))
model.compile(
 loss='categorical_crossentropy',
 optimizer='adam',
 metrics=['acc']
)
history = model.fit(
   X_train, y_train,
   epochs=50,
   batch_size=32,
   validation_split=0.1,
   shuffle=False
)
def plot_learningCurve(history, epochs):
 epoch_range = range(1, epochs+1)
 plt.plot(epoch_range, history.history['loss'])
 plt.plot(epoch_range, history.history['val_loss'])
 plt.title('Model loss')
 plt.ylabel('Loss')
 plt.xlabel('Epoch')
 plt.legend(['Train', 'Val'], loc='upper left')
 plt.show()
plot_learningCurve(history, 50)
pip install mlxtend
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import plot_confusion_matrix
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
y_pred.shape,y_test.shape,X_test.shape
from sklearn.metrics import confusion_matrix

def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  cm=cm/ cm.astype(np.float).sum(axis=1)
  fig, ax = plt.subplots(figsize=(7, 7))
  ax = sns.heatmap(
     cm,
     annot=True,
     ax=ax,
     cmap="OrRd"
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.title('Matrice de confusion normalisée')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  plt.show()
plot_cm(
  enc.inverse_transform(y_test),
  enc.inverse_transform(y_pred),label.classes_

)
def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  fig, ax = plt.subplots(figsize=(7, 7))
  ax = sns.heatmap(
     cm,
     annot=True,
     ax=ax,
     cmap="OrRd",
     fmt='g'
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  plt.title('Matrice de confusion non normalisée')
```

```
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  plt.show()
plot_cm(
  enc.inverse_transform(y_test),
  enc.inverse_transform(y_pred),label.classes_

)
```

Ce code est inspiré du document Time Series Classification for Human Activity Recognition with LSTMs in Keras. Dans : *Curiousily* [en ligne]. [s. d.]. [Consulté le 17 mai 2021]. Disponible à l'adresse : https://curiousily.com/posts/time-series-classification-for-human-activity-recognition-with-lstms-in-keras/

TABLE DES FIGURES

TABLE DES MATIERES