UNIVERSITÉ DE NANTES

IAE NANTES
ÉCONOMIE & MANAGEMENT

**Master 2 Économétrie et Statistiques, parcours Économétrie Appliquée**

# SVM et réseaux de neurones

*Gabin Lagre et Astrid Valicon*

Année 2021/2022

EKAP
Master Économétrie Appliquée

Sommaire

# I. Introduction

La découverte des ondes gravitationnelles est une révolution pour le domaine de la science. Il s'agit de signaux émis par la collision de deux trous noirs. Cette ondulation est causée par des par des processus très violents et énergétique de l'Univers. Une onde gravitationnelle s'apparente à une ondulation dans l'espace-temps, ces signaux sont minuscules. Les meilleurs instruments de détection de la planète sont mis à contribution et les signaux observés sont enfouis dans le bruit des détecteurs. Nous utiliserons G2NET qui est un réseau d'ondes gravitationnelles, de géophysique et d'apprentissage automatique.

L'objectif de notre compétition Kaggle est de détecter les signaux des ondes gravitationnelles. Nous allons créer des modèles pour analyser les données de série chronologiques des ondes gravitationnelles qui ont été simulées à partir d'un réseau de détecteurs terrestres. Quatre fichiers, que nous détaillerons plus loin, sont à notre disposition. L'ensemble du projet sera réalisé grâce au langage de programmation Python.

Nous appliquerons des techniques de Machine Learning. Il s'agit d'une application de l'intelligence artificielle qui propose aux systèmes informatiques la possibilité d'apprendre et de s'améliorer automatiquement à partir d'une expérience effectuée au préalable. Cette méthode s'appuie sur le développement de programmes informatiques qui peuvent accéder aux données et les utiliser pour apprendre d'eux-mêmes, afin d'être appliquées sur des données inédites. Le Machine Learning permet aux algorithmes d'apprendre automatiquement sans intervention humaine ni assistance, et d'ajuster les actions en conséquence. Il existe plusieurs méthodes d'apprentissage automatique : l'apprentissage supervisé et l'apprentissage non-supervisé. Dans notre projet, nous utiliserons l'apprentissage supervisé pour répondre à la problématique.

Dans un premier temps nous présenterons et préparerons les données complexes mise à notre disposition. Dans un second temps nous présenterons la méthodologie utilisée. Puis, nous appliquerons les modélisations sur nos jeux de données. Enfin, nous concluerons et discuterons des résultats obtenus.

# II. Présentation et préparation des données

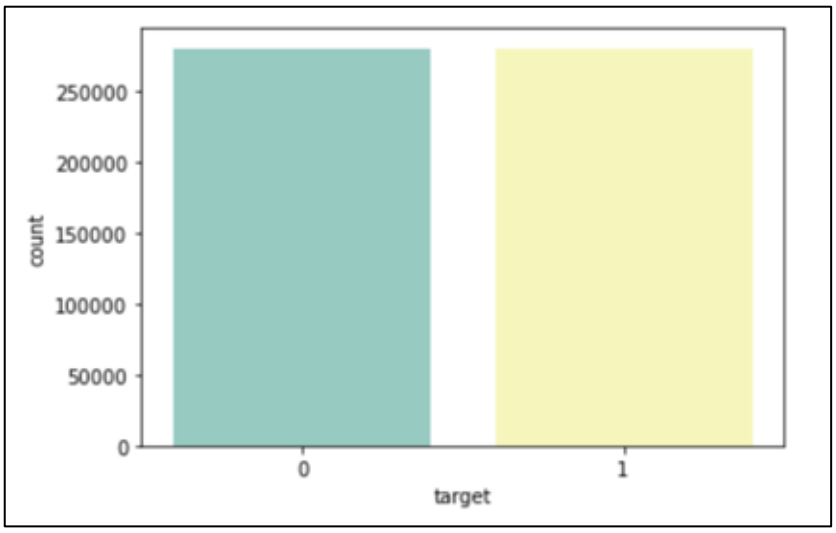
Nous avons à notre disposition quatre fichiers. Les deux premiers sont des dossiers comprenant des fichiers .npy qui regroupent eux même des séries chronologiques. Ces dernières contiennent des mesures simulées d'ondes gravitationnelles à partir d'un réseau de trois interféromètres à ondes gravitationnelle. Il s'agit de LIGO Hanford, LIGO Livingston et Virgo. Chacune des séries temporelles (une pour chaque détecteur) contiennent soit le bruit du détecteur, soit le bruit du détecteur plus un signal d'onde gravitationnelle simulé. Nous précisons également que chaque cycle dure deux secondes est échantillonné à 2048 Hz. Notre tâche consiste à déterminer quand le signal est présent dans les données (lorsque « target » = 1). Cette problématique consiste en l'application d'une classification binaire, si le signal est détecté ou non. Le fichier « train » est le fichier sur lequel nous allons travailler. Le fichier « test » nous permettra de comparer nos prévisions.

Il existe plusieurs variables pouvant impacter la forme des ondes gravitationnelles. Il s'agit de la masse, l'emplacement du ciel, la distance, les spins de trou noir, l'angle d'orientation binaire, la polarisation gravitationnelle d'onde, l'heure d'arrivée et la phase à coalescence (fusion). Ces paramètres ne sont pas intégrés dans notre jeu de données mais ont été randomisés et utilisés pour générer les signaux présents dans la base.

Les deux autres fichiers sont des fichiers .csv. L'un correspond aux valeurs cibles indiquant si le signal associé contient une onde gravitationnelle (target = 1) ou non (target = 0). Ce sont les données dites d'apprentissage. L'autre correspond aux données test, qui nous permettra de vérifier à la fin nos résultats et les comparer.

Nous commençons par importer nos données d'apprentissage en format .csv (que nous appelons « train_label »), puis en format .npy (que nous appelons « train_path »).

***Train_label*** : Cette base comporte 560000 observations et 2 colonnes. La première colonne correspond à l'identifiant (id) et la seconde correspond à la valeur cible, qui prend la valeur 0 ou 1. Il n'y a pas de valeurs nulles. Grâce à la commande value_counts(), nous pouvons voir que 280070 observations prennent la valeur 0 et 279930 prennent la valeur 1 *(cf graphique 1)*. Il y a donc au total, 279930 signaux d'ondes gravitationnelles dans la base de données, ce qui représente 49,98% des données.

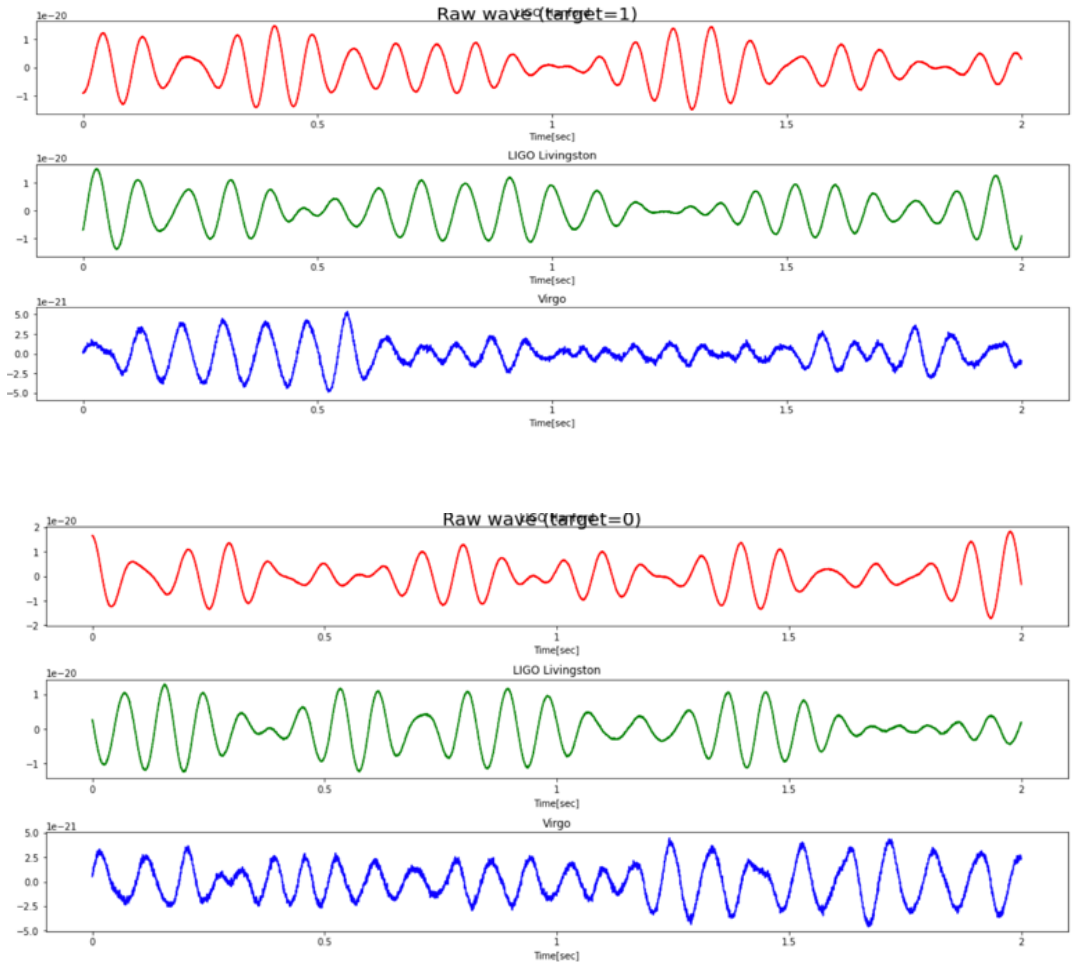



*Graphique 1 : Répartition des cibles = 0 et cibles = 1 dans nos données*

***Train_path*** : Ces données ont été importées grâce à la librairie glob. Il y a 560 000 fichiers .npy dans ce dossier. Nous en choisissons un pour l'observer. L'échantillon comporte 3 lignes car ce sont les séries chronologiques de 3 interféromètres à ondes gravitationnelles évoqués plus haut. Il y a 4096 colonnes. Ces chiffres sont les mêmes pour l'ensemble des fichiers .npy. Chaque fichier .npy (comprenant 3 lignes et 4096 colonnes) et assigné à une valeur cible (soit 1 soit 0).

Nous fusionnons le fichier « train_label » avec le fichier « train_path » à l'aide la fonction merge. Nous obtenons ainsi une base « df » comprenant 3 colonnes avec les variables id, path et target et 560 000 lignes.

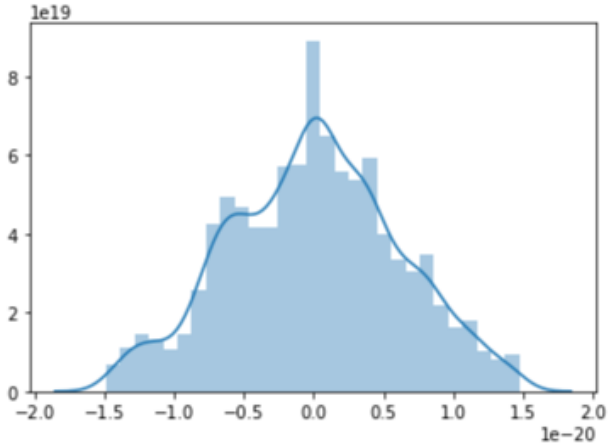
Nous réalisons plusieurs commandes sur Python afin de découvrir et visualiser notre jeu de données. Ci-dessous se trouvent des graphiques représentant les signaux des 3 interféromètres différents, selon target = 1 et target = 0.




*Graphique 2 : Signaux des différents interféromètres selon si target = 1 ou target = 0*

Lorsqu'il y a la présence d'une onde (target=1), le signal est modifié. En effet, sur les trois premiers graphiques, nous observons une amplification ou un signal très affaibli. Dans le cas contraire, sur les trois graphiques d'après, nous retrouvons le signal classique de l'onde gravitationnelle reçu en permanence.

Nous observons également la distribution de notre échantillon.




*Graphique 3 : Distribution de notre échantillon*

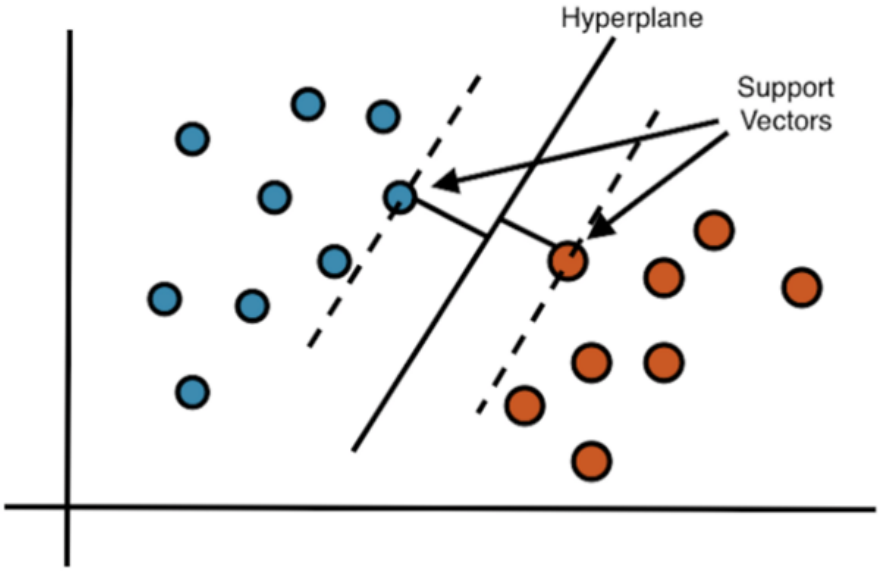
Pour la suite de l'étude, nous nous appuyons sur un code en particulier (https://github.com/SiddharthPatel45/gravitational-wave-detection/blob/main/code/gw-detection-modelling.ipynb). Notre jeu de données est très important et son traitement peut créer un goulot d'étranglement sur l'ensemble du flux de travail. Cela peut causer une surcharge de mémoire. Le code sur lequel nous nous appuyons propose un moyen de réduire notre jeu de données en créant « un jeu de données d'entrée » grâce à la librairie TensorFlow. TensorFlow est une bibliothèque open source de Machine Learning, créée par Google. C'est une boîte à outils qui permet de résoudre des problèmes mathématiques complexes avec plus de facilité. Elle permet notamment d'exécuter et d'entraîner des réseaux de neurones, la reconnaissance d'image, les plongements de mots… Elle nous sera donc très utile pour notre projet. C'est un système de programmation dans lequel les calculs sont représentés par des graphiques.

Nous souhaitons désormais séparer notre base de données en deux : un jeu de test et un jeu d'apprentissage avec la fonction train_test_split. Cette fonction se trouve dans la bibliothèque Sickit-Learn, qui est destinée au Machine Learning et à la data science avec Python. Cette bibliothèque implémente de nombreux algorithmes tels que les réseaux de neurones (perceptron multicouches), les SVM, les k plus proches voisins… Nous pouvons donc entraîner plusieurs modèles et mesurer leur performance.

# III. Méthodologie utilisée

## 1. SVM

Les SVM (Support Vector Machines) correspondent à un ensemble de méthodes d'apprentissage supervisé utilisées pour la classification, la régression et la détection des valeurs aberrantes. Le plus souvent, ils sont utilisés pour la classification, notamment la classification binaire. Ils ont été développés dans les années 1990. L'apprentissage supervisé est utilisé lorsque l'on dispose d'un échantillon pour lequel on connaît les valeurs que doit avoir la sortie du réseau en fonction des entrées. Le réseau apprend à partir de cet échantillon. Les SVM ont pour objectif de séparer les données en classes à l'aide d'une frontière, de telle sorte que la distance entre les groupes de données et la frontière qui les sépare soit maximale. Cette frontière est appelée « la marge ». L'hyperplan optimal est l'hyperplan à marge maximale. La maximisation de la marge permet une classification plus précise et donc de meilleurs résultats. Cette marge suppose que les données soient linéairement séparables, ce qui est rarement le cas. C'est pour cette raison que les SVM reposent souvent sur l'utilisation de « noyaux ». Nous distinguons donc les SVM linéaires des SVM non linéaires.




Graphique n°4 : Hyperplan divisant les données en 2 classes.

Dans le cas où les données ne sont pas linéairement séparables, nous pouvons changer et augmenter la dimension de notre hyperplan dans lequel il y a probablement un séparateur linéaire. C'est donc dans ce cas précis que la fonction noyau est nécessaire. Il repose sur le théorème de Mercer.

## 2. Réseaux de neurones (ANN et CNN)

Les réseaux de neurones sont issus de l'Intelligence artificielle. La motivation sous-jacente est de créer des modèles qui imitent le raisonnement humain (comment l'homme fait-il pour raisonner, calculer, apprendre ?). Cette imitation permet donc de résoudre des problématiques de Machine Learning. Les domaines d'application sont nombreux : reconnaissance d'image, classifications de texte ou d'images, identification d'objets, prédiction de données…Les architectures de réseaux de neurones sont divisées en plusieurs familles. Les voici :

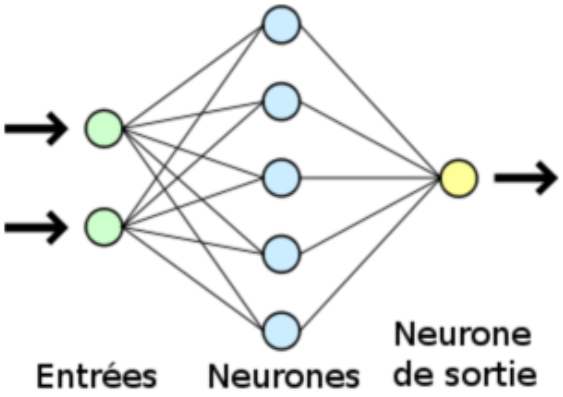
- Réseaux de neurones Feed-forwarded
- Réseaux de neurones récurrent (RNN)
- Réseaux de neurones à résonance
- Réseaux de neurones auto-organisés

### a) Réseaux de neurones artificiels (ANN)

Pour notre projet, nous appliquerons les réseaux de neurones Feed-forward (propagation avant), qui sont ceux que nous avons vu en cours et que nous maîtrisons le mieux. La « propagation avant » signifie que la donnée traverse le réseau d'entrée jusqu'à la sortie sans retour en arrière de l'information. Dans cette même famille, nous distinguons

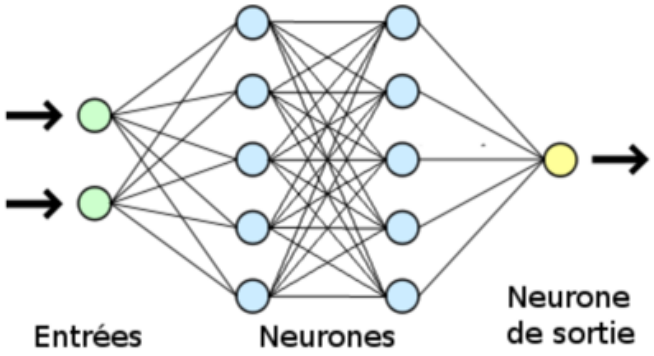
- Les réseaux monocouches (perceptron simple)
- Les réseaux multicouches (perceptron multicouche)

Le perceptron simple ne dispose que de deux couches, la couche en entrée et la couche en sortie. Le réseau se déclenche lorsqu'il reçoit une information en entrée. Ensuite, l'information et la donnée sont traitées entre la couche d'entrée et celle de sorties. Elles sont toutes reliées entre elle. Le réseau entier ne dispose que d'une matrice de poids, ce qui limite le perceptron simple à un classificateur linéaire. Il permet alors de diviser toutes les informations obtenues en deux catégories distinctes. Ci-dessous est représenté le perceptron simple sous forme de schéma afin de mieux visualiser l'algorithme.




Graphique n°5 : Réseau de neurones (perceptron simple)

Le perceptron multicouche est structuré de la même façon que le perceptron simple. L'information rentre par une couche d'entrée et sort par la couche de sortie. Ce qui diffère du perceptron simple, c'est qu'en la couche d'entrée et la couche de sortie, il existe une ou plusieurs couches dites « cachées ». Il y a autant de couches qu'il y a de matrices de poids dans le réseau. Il est souvent utilisé pour traiter les fonctions de type non-linéaires. Ci-dessous, la représentation d'un perceptron multicouches.




Graphique n°6 : Réseau de neurones (perceptron multicouche)

### b) Réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutif permettent le traitement d'informations complexes et très variées. Il s'agit de créer plusieurs réseaux de neurones distincts afin de traiter chacun une partie de l'information. Ces réseaux peuvent être imaginés comme une compilation d'un segment

d'informations pour, finalement, traiter l'ensemble de l'information. Ils sont souvent utilisés pour le traitement d'image, de vidéos, de textes).

# I. Application et modélisation

Dans notre cas, il s'agit ici d'un problème de traitement de signal avec la tâche de classification.

### a) Prétraitement des données et création du modèle

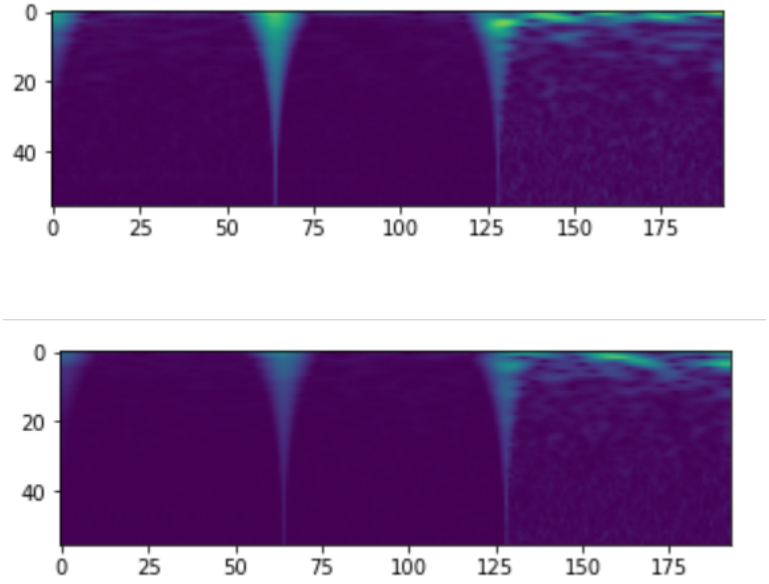
Si une fréquence à un signal qui diffère, c'est-à-dire la suppression ou une plus forte amplitude du signal alors nous estimerons qu'il y a la présence d'une onde gravitationnelle.

Nous commençons par transformer le signal original en spectrogramme (en image). La constant Q transformation (CQT), permet de transformer une série de données dans le domaine fréquentiel. Dans notre cas nous utiliserons le package nnAudio pour traiter nos fichier numpy sous forme de fréquence. Nous rappelons qu'il y a trois signaux pour chaque interféromètre.

Afin d'éviter un effet de surcharge de la mémoire nous créons une sorte de pipeline. Pour cela nous utilisons trensorflow (TF). L'avantage de TF est qu'il peut être installé soit pour utiliser le GPU (partie processeur) ou le CPU (carte graphique), cela permet d'être dans des conditions optimales en fonction de notre matériel. Ce pipeline est indispensable car nous avons besoin de faire un pré-traitement des données pour la passer sous forme de spectrogramme. En effet, il convient de n'utiliser qu'une partie seulement des données pour nos modèles. C'est pour cette raison que le prétraitement est nécessaire.

Nous devons donc définir certains paramètres des signaux en tenant compte du fait que les données fournies sont à 2048 Hz et que chaque signal dure 2 secondes. Nous appliquons aussi les paramètres du modèle comme la taille du batch, qui correspond au nombre d'images utilisées pour entraîner le réseau (nous appliquons 250). Nous tenons compte également du nombre d'epochs qui correspond à un apprentissage sur toutes les données. Plus ce nombre est grand et plus nous devrions obtenir une bonne précision, mais le modèle est donc plus long à tourner. Dans notre cas, étant donné que notre jeu de données est assez important, nous ajoutons 3 epochs.

Nous transformons ensuite nos données sous format. npy en images (spectrogrammes). Un spectrogramme est un diagramme représentant le spectre d'un phénomène périodique, associant à chaque fréquence une intensité ou une puissance. Nous pouvons visualiser deux de ces images ci-dessous, de dimensions (59, 193, 1).




*Graphique n°7 : Spectrogrammes*

Nous créons ensuite notre jeu de validation et d'entrainement à l'aide d'un train_test_split

Pour la création de notre modèle nous utilisons les réseaux de neurones convolutif (CNN), c'est un algorithme du Deep Learning. Nous avons donc un empilage multicouche de perceptrons avec comme objectif de prétraiter des petites quantités d'information.

Concernant l'évaluation du modèle, nous allons utiliser le critère AUC (Area under ROC curve) et la courbe ROC (receiver operating characteristic curve) qui nous indiquent si le modèle est bon pour séparer les deux classes. En effet, l'AUC représente l'air sous la courbe ROC. Plus l'AUC se rapproche de 1 et plus le modèle est bon pour séparer les deux classes. Nous regarderons aussi la précision de notre modèle, ce qui nous donnera une idée de la performance globale.

Nous estimons notre modèle CNN de façon séquentiel. Les résultats se trouvent ci-dessous.

```
Model: "CNN_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv_01 (Conv2D)             (None, 54, 191, 16)       160
_________________________________________________________________
Pool_01 (MaxPooling2D)       (None, 27, 95, 16)        0
_________________________________________________________________
Conv_02 (Conv2D)             (None, 25, 93, 32)        4640
_________________________________________________________________
Pool_02 (MaxPooling2D)       (None, 12, 46, 32)        0
_________________________________________________________________
Conv_03 (Conv2D)             (None, 10, 44, 64)        18496
_________________________________________________________________
Pool_03 (MaxPooling2D)       (None, 5, 22, 64)         0
_________________________________________________________________
Flatten (Flatten)            (None, 7040)              0
_________________________________________________________________
Dense_01 (Dense)             (None, 512)               3604992
_________________________________________________________________
Dense_02 (Dense)             (None, 64)                32832
_________________________________________________________________
Output (Dense)               (None, 1)                 65
=================================================================
Total params: 3,661,185
Trainable params: 3,661,185
Non-trainable params: 0
```

*Tableau n°1 : Architecture de notre modèle CNN*

Ci-dessus, nous retrouvons l'architecture de notre modèle avec les différentes séquences, les tailles des images ainsi que les paramètres associés à chaque image.

*Tableau n°2 : Résultat de notre modèle CNN sur les données d'apprentissage*

```
Epoch 1/3
1680/1680 [==============================] - 5244s 3s/step - loss: 0.4739 - auc: 0.8331 - accuracy: 0.7623 - val_loss: 0.4674 -
val_auc: 0.8400 - val_accuracy: 0.7667
Epoch 2/3
1680/1680 [==============================] - 5482s 3s/step - loss: 0.4705 - auc: 0.8353 - accuracy: 0.7642 - val_loss: 0.4765 -
val_auc: 0.8399 - val_accuracy: 0.7599
Epoch 3/3
1680/1680 [==============================] - 5463s 3s/step - loss: 0.4688 - auc: 0.8362 - accuracy: 0.7650 - val_loss: 0.4601 -
val_auc: 0.8431 - val_accuracy: 0.7709
```

Le modèle CNN estimé est assez long à être estimer. En effet, il lui faut environ 4 heures pour sortir des résultats. Le modèle peut donc surement être amélioré avec une formation plus poussée de la structure. Nous obtenons finalement un AUC de 0,836 et une précision (accuracy) de 0,765 tandis que pour les données de validation (ci-dessous), nous obtenons un AUC de 0,842 et une précision de 0,77. L'AUC indique donc que notre modèle est bon pour séparer nos deux classes : target = 1 et target = 0 (présence d'une onde ou non).

*Tableau n°3 : Résultat de notre modèle CNN sur les données de validation*

```
Epoch 1/3
560/560 [==============================] - 1592s 3s/step - loss: 0.4632 - auc: 0.8404 - accuracy: 0.7683
Epoch 2/3
560/560 [==============================] - 2216s 4s/step - loss: 0.4617 - auc: 0.8415 - accuracy: 0.7694
Epoch 3/3
560/560 [==============================] - 1414s 3s/step - loss: 0.4605 - auc: 0.8423 - accuracy: 0.7703
```

### b) Prédictions

Nous pouvons désormais faire nos prédictions sur le jeu de données test. Voici les prédictions obtenues sur le jeu de données test pour notre valeur cible. Nous obtenons des chiffres cohérents.

*Tableau n°4 : Prédiction de la target en fonction de l'id*

| | id | target |
|---|---|---|
| 0 | 00005bced6 | 0.999814 |
| 1 | 0000806717 | 0.688297 |
| 2 | 0000ef4fe1 | 0.288745 |
| 3 | 00020de251 | 0.476189 |
| 4 | 00024887b5 | 0.226742 |

Nous avons établi un tableau qui récapitule l'ensemble des résultats de notre modèle CNN. Le temps d'exécution était long dû a des problèmes de train_test_split entre la version Mac OS et Windows 11. Nous avons donc choisi de faire un train_test_split sur toute la base de données. Ainsi, nous n'avons pas eu de soucis avec cette commande. Nous obtenons de bons résultats même si la mise en place du modèle reste à améliorer. Les codes déjà présents sur Kaggle et Github nous ont été d'une aide précieuse.

*Tableau n°5 : Récapitulatif de nos résultats*

| | Temps d'exécution | AUC (jeu d'apprentissage) | AUC (jeu de validation) | Précision (jeu d'apprentissage) | Précision (jeu de validation) | Test AUC Kaggle |
|---|---|---|---|---|---|---|
| Modèle CNN | **~ 4h30** | **0,836** | **0,842** | **0,765** | **0,77** | **0,845** |

# II. Conclusion et discussion

L'objectif de ce projet Kaggle était de repérer la présence d'ondes gravitationnelles à travers des signaux calculés par trois interféromètres différents. Il s'agit de LIGO Hanford, LIGO Livingston et Virgo. Pour cela, nous avons transformé ces signaux en images, en spectrogrammes.

Nous avons pu observer que les ondes gravitationnelles sont difficiles à détecter. Après avoir essayé de faire un prétraitement sur la base de données et transformer les données en spectrogrammes, nous avons formé un modèle d'apprentissage. Il s'agit de réseaux de neurones convolutifs. Les données étaient très nombreuses ce qui a rendu les estimations assez complexes. En effet, le modèle a été long à estimer sur Python (4h30 environ).

Finalement, le modèle a été relativement performant avec un AUC de 0,83 et une précision de 0,76. L'AUC est proche de 1 ce qui signifie que le modèle est bon pour séparer les deux classes : la présence d'une onde gravitationnelle dans le signal ou non.

Cette compétition Kaggle était complexe et un peu au-dessus de nos compétences. Malgré cela, nous avons pu découvrir une technique en Deep Learning permettant le traitement d'informations complexes. Ici, il s'agissait du traitement des images. Cette étude nous a permis d'élargir nos connaissances en Machine Learning, notamment sur les réseaux de neurones.

## Bibliographie :

Réseaux de neurones : https://www.juripredis.com/fr/blog/id-19-demystifier-le-machine-learning-partie-2-les-reseaux-de-neurones-artificiels

Machine Learning : https://www.oracle.com/fr/data-science/machine-learning/what-is-machine-learning/

TensorFlow : https://www.lebigdata.fr/tensorflow-definition-tout-savoir

https://www.kaggle.com/lakshit28/g2net-gravitational-wave-detection

https://github.com/SiddharthPatel45/gravitational-wave-detection/blob/main/code/gw-detection-modelling.ipynb

https://www.kaggle.com/atamazian/nnaudio-constant-q-transform-demonstration/comments