ISEN
MÉDITERRANÉE

2025

RAPPORT PROJET IA/JAVA/THS

arthur.delhaise-ramond@isen.yncrea.fr
anas.sadik@isen.yncrea.fr
lucie.ribero@isen.yncrea.fr
tom.riquier@isen.yncrea.fr

Anas, Lucie, Arthur, Tom

SOMMAIRE

01 INTRODUCTION

CONTEXTE DU PROJET

Dans le cadre de notre formation d'ingénieur à l'ISEN Méditerranée, nous avons dû étudier ce projet : il s'appuie sur plusieurs de nos matières à savoir le traitement du signal (THS), intelligence artificielle (IA) ou encore la programmation orientée objet (POO) en Java.
Notre but est de créer un programme Java permettant de distinguer automatiquement si un fichier audio contient un aboiement ou un miaulement.

Nous avons utilisé deux techniques importantes pour notre projet :

- La transformée de Fourier rapide (FFT) pour convertir le son dans le domaine des fréquences.
- Un neurone artificiel pour apprendre à classer les sons selon leurs caractéristiques spectrales.

OBJECTIFS PÉDAGOGIQUES ET TECHNIQUES

Ainsi, ce projet nous a permis d'acquérir plusieurs compétences, et d'atteindre plusieurs objectifs, à savoir :

- Appliquer de façon concrète des notions vues lors du cours de traitement du signal.
- Comprendre et manipuler de « simple » neurones artificiels.
- Relier la théorie et l'application à travers une démarche expérimentale.
- D'analyser des résultats, en tirer des conclusions, et finalement proposer des améliorations.
- Lire et traiter des fichiers audio au format WAV
- Implémenter une chaîne de traitement complète en Java
- Tester la robustesse d'un système face au bruit et à des données inconnues
- Comparer 3 fonctions d'activation de neurones et fournir une implémentation en Java. Les fonctionnalités devraient inclure Heaviside, Sigmoïde et ReLU.

OBJECTIFS PÉDAGOGIQUES ET TECHNIQUES

Nous avons divisé notre réflexion en 4 axes : tester la FFT sur des signaux simples (cos, sin) afin de bien comprendre le passage en domaine fréquentiel, prendre en main les neurones artificiels sur des fonctions logiques types ET, OU, assembler la chaîne complète pour classifier des sons réels, en extrayant des vecteurs FFT et enfin, tester notre système sur des sons inconnus et comparer les résultats selon les fonctions d'activation.
À chaque étape, nous avons pris soin de comprendre ce que nous faisions, de documenter nos observations et de proposer des améliorations.

FONDEMENTS THÉORIQUES

02

TRANSFORMÉE DE FOURIER RAPIDE (FFT)

PRINCIPE ET UTILITÉ

La transformée de Fourier est une méthode permettant de convertir un signal dans le temps c'est-à-dire une onde sonore, en un signal dans le domaine des fréquences. En d'autres termes, cela permet de déterminer les fréquences qui composent un son. Par exemple, un aboiement a des fréquences totalement différentes de celles d'un miaulement. Il s'agit d'un élément fondamental de notre implémentation, car nous utiliserons des fichiers audio pour obtenir des informations utiles.

IMPLÉMENTATION JAVA ET TEST SUR SIGNAUX SIMPLES

Nous avons utilisé le fichier FFTCplx.java fourni, qui implémente l'algorithme de Cooley-Tukey. Pour tester le fonctionnement, on a appliqué la FFT à des signaux simples créés nous-mêmes, comme :

- un cosinus pur avec une seule période,
- puis avec plusieurs périodes (jusqu'à 8),
- puis un sinus (équivalent mais déphasé de π/2),
- enfin, en plaçant le signal sur l'axe imaginaire (pour observer l'effet sur la phase et l'énergie spectrale).

ANALYSE DES RÉSULTATS ET INTERPRÉTATIONS

- Avec NbPériodes = 1, on observe un pic de module au coefficient k = 1, et par symétrie à k = N - 1. Cela confirme que la FFT détecte bien la fréquence dominante.
- Si on augmente la fréquence (NbPériodes entre 1 et 7), le pic se déplace vers la droite (valeur de k plus grande).
- À NbPériodes = 8, on atteint la fréquence de Nyquist, et un seul pic apparaît à k = 8.
- En passant de cos à sin, les pics restent aux mêmes endroits mais la phase change (décalage dû à π/2).
- Quand on place le signal sur l'axe imaginaire, les parties réelle et imaginaire du spectre sont inversées.

Conclusion : La FFT est fonctionnelle pour déterminer la fréquence dominante d'un signal. Les pics ou la phase changent selon le type de signal mais pas l'amplitude.

TRANSFORMÉE DE FOURIER RAPIDE (FFT)

ARCHITECTURE D'UN NEURONE

Un neurone artificiel imite le fonctionnement d'un neurone biologique. Il reçoit N entrées, les multiplie par un poids puis les somme et ajoute un biais au total. Enfin, on applique une fonction d'activation à la somme des entrées. Le résultat est la sortie du neurone, que l'on peut considérer comme une prédiction.

FONCTIONS D'ACTIVATION (HEAVISIDE, SIGMOÏDE, RELU)

- Heaviside : donne 0 si la somme est < 0, sinon 1. C'est une fonction brutale, à seuil fixe.
- Sigmoïde : donne un résultat progressif entre 0 et 1. Elle est plus douce et permet une meilleure gestion du bruit.
- ReLU : donne 0 si la somme est négative, et la valeur brute sinon. Elle est très utilisée en apprentissage moderne.

GAMME DE FRÉQUENCES ANALYSABLE

La transformée de Fourier rapide (FFT) permet d'analyser les fréquences contenues dans un signal échantillonné. La gamme des fréquences analysables dépend de la taille de la FFT, c'est-à-dire du nombre d'échantillons N, et de la durée d sur laquelle ces échantillons ont été prélevés. La fréquence d'échantillonnage Fe, qui correspond au nombre d'échantillons par seconde, est égale au nombre d'échantillons divisé par la durée : $Fe=N/d$. La FFT peut analyser les fréquences de 0 Hz jusqu'à la moitié de cette fréquence d'échantillonnage, appelée fréquence de Nyquist, soit $f_{max} = Fe / 2 = N / (2d)$. Enfin, la résolution fréquentielle, c'est-à-dire la distance entre deux fréquences consécutives analysées, est égale à $\Delta f = 1 / d$.

ALGORITHME D'APPRENTISSAGE

Le neurone apprend à prédire une sortie correcte en ajustant ses poids selon l'erreur entre la sortie obtenue et la sortie attendue. Cela se fait en minimisant une fonction de coût (ici, l'erreur quadratique moyenne, MSE). L'apprentissage se fait par itérations successives jusqu'à ce que l'erreur soit suffisamment petite.

TESTS SUR FONCTIONS LOGIQUES (ET / OU)

Nous avons entraîné un neurone Heaviside sur deux fonctions logiques simples :

- Pour ET : [0,0,0,1] → il faut que les deux entrées soient 1 pour activer le neurone.
- Pour OU : [0,1,1,1] → il suffit qu'une seule entrée soit 1.

Dans les deux cas, le neurone a réussi à apprendre. Cela montre qu'il peut gérer les cas linéairement séparables.

Par contre, il n'a pas réussi avec XOR, car cette fonction n'est pas séparable par une seule droite (il faudrait un réseau plus complexe).

Observation :

- Pour la fonction ET, les poids sont faibles, mais le biais est fort négatif, pour que seule l'entrée (1,1) déclenche la sortie 1.
- Pour la fonction OU, les poids sont plus forts, car il suffit qu'un seul soit élevé pour activer la sortie.

ROBUSTESSE FACE AU BRUIT

Pour simuler une situation réaliste dans laquelle les entrées ne sont pas parfaites, nous avons rajouté un bruit gaussien de moyenne nulle et d'amplitude choisie allant de 0.1 (bruit faible) à 0.6 (bruit fort) sur les valeurs binaires attendues (valeurs 0 ou 1). Cela a pour but de voir si le neurone est capable de produire à nouveau la réponse adéquate même en présence de bruit.

Fonction ET avec bruit :

Lorsque l'on rajoute du bruit sur nos entrées pour simuler un neurone ET, les entrées évoluent selon la fonction d'activation utilisée :

- Pour la fonction Heaviside, le premier test nous rend bien les entrées souhaitées. Cependant, au bout de 3 essais supplémentaires, les entrées sont erronées faisant notamment passer la dernière entrée, censé être à 1 à 0.
- Pour la fonction ReLU, les résultats des entrées est approximativement bien (pour les 3 premières environ 0 et pour la dernière environ 1). Au bout de 4-5 essais, certaines entrées commencent à s'éloigner de leur valeur souhaitée.
- Pour la fonction sigmoide, au bout de 6-7 essais, les entrées ne changent quasiment pas, témoignant d'une robustesse plus importante que les deux autres fonctions d'activation.

Fonction OU avec bruit :

- Le neurone Heaviside se trompe fréquemment, environ une fois sur deux, à cause de son seuil rigide et du passage brusque entre 0 et 1. Le bruit rend l'interprétation des entrées aléatoire.

- On constate que la fonction ReLU est incapable de converger correctement car elle finit toujours avec une erreur quadratique moyenne bien trop élevée (environ 0.62) et elle ne parvient pas à stabiliser ses 4 entrées. Le neurone reste donc coincé dans une boucle d'apprentissage sans aucune amélioration.
- La fonction sigmoide parvient à garder des valeurs proches et justes au bout de plusieurs essais successifs en montrant de ce fait une meilleure faculté à s'adapter à des données bruitées.

Conclusion générale :
Le neurone Heaviside est rapide et simple, mais il est vulnérable au bruit car il donne systématiquement une sortie binaire. La fonction ReLU permet de contrecarrer ce phénomène mais on remarque qu'elle est un peu moins stable. Enfin, on comprend maintenant pourquoi la sigmoïde est assez lente à apprendre : elle est beaucoup plus robuste et fiable que les autres fonctions lorsque les données sont bruitées.
Ces observations expliquent pourquoi, dans des applications pratiques, on privilégie souvent des fonctions d'activation comme la sigmoïde ou ReLU pour traiter des données réelles incertaines.La FFT avec un neurone simple marche bien mais Heaviside est sensible au bruit. Sigmoïde et ReLU sont plus robustes. Pour améliorer, il faut utiliser des réseaux plus complexes, mieux prétraiter le son, et augmenter les données.

IMPLÉMENTATION DE LA CHAÎNE DE TRAITEMENT

03

LECTURE ET DÉCOUPAGE DES FICHIERS AUDIO

Nous avons utilisé des fichiers .wav pour l'entraînement et les tests. Dans ces enregistrements, on trouve soit des aboiements, soit des miaulements.
Le fichier Son.java que l'on nous a fourni nous a bien rendu service :

- Il a chargé pour nous les échantillons sonores en mémoire dans un tableau de floats.
- Il a ensuite découpé chaque enregistrement n'importe quelle longueur en morceaux de taille fixe (par exemple 1024 échantillons chacun).
- Cela va nous permettre d'analyser indépendamment chacun de ces petits extraits de son comme si notre technique s'appliquait à un seul grand nombre d'enregistrements.

Nous pouvons ainsi nous poser la question : pourquoi devons nous découper en morceaux ?
Il faut choisir la bonne taille de morceau, car il y a un compromis à faire : un "morceau" très petit ne donnera pas d'informations très précises en fréquence, en revanche un "morceau" très long contiendra peut-être plusieurs sons différents.
Avec une fréquence d'échantillonnage de 44100 Hz (classique), 1024 points correspondent à environ 23 ms, ce qui nous donne une bonne résolution en fréquence (résolution $\approx$ 43 Hz) tout en nous permettant de bien capturer l'aboiement ou le miaulement correspondant. C'est donc un choix raisonnable et adapté à notre problème.

PASSAGE DANS LE DOMAINE FRÉQUENTIEL

Une fois les blocs récupérés, on a calculé la transformée de Fourier de chaque bloc avec la méthode FFTCplx.appliqueSur(). On obtient pour chaque bloc un tableau de nombres complexes, représentant les fréquences contenues dans le son.

Pour chaque fréquence, on a calculé le module, car c'est lui que l'on va utiliser pour entraîner le classifieur. Le module nous indique à quel point on retrouve une certaine fréquence dans le bloc.

Pour chaque bloc, on ne garde que les 20 premières valeurs de module, c'est-à-dire les fréquences graves, parce qu'elles contiennent la majorité de l'information permettant de distinguer les sons vocaux comme les miaulements ou les aboiements.

Comparaison des graphes FFT en Annexe 1, 2, 3

CONSTITUTION DES VECTEURS D'ENTRÉE

Pour chaque bloc, nous avons donc maintenant 20 valeurs représentant les amplitudes des premières fréquences.

Ces valeurs sont normalisées entre 0 et 1 en divisant par la valeur maximale du bloc.
Cela donne un vecteur d'entrée de 20 nombres, utilisable directement par le neurone.

Chaque bloc audio devient donc un vecteur (entrée), avec une étiquette associée :

- 1 si le bloc vient d'un fichier « miaulement »
- 0 s'il vient d'un fichier « aboiement »

ENTRAÎNEMENT DU NEURONE

Nous avons entraîné un neurone artificiel à reconnaître si un vecteur correspond à un chat ou à un chien. L'entraînement consiste à lui montrer plusieurs exemples (vecteurs + étiquettes), et à ajuster les poids internes du neurone pour qu'il donne la bonne réponse le plus souvent possible.

DONNÉES D'ENTRAÎNEMENT

Nous avons utilisé deux fichiers audios :

- Un fichier contenant uniquement des miaulements
- Un autre contenant uniquement des aboiements

Nous avons extrait 20 blocs par classe, soit 40 blocs au total pour entraîner le neurone. Cela représente un petit jeu de données, mais suffisant pour tester les performances de base du neurone.

Voir en Annexe 4 la fonction apprentissage() qui permet d'entrer le neurone.

COURBE D'APPRENTISSAGE (MSE)

Pendant l'apprentissage, on suit la valeur de l'erreur quadratique moyenne (MSE). Elle mesure l'écart entre la sortie du neurone et la valeur attendue.

En théorie :

- Si le MSE est grand, le neurone se trompe souvent.
- Si le MSE devient petit (proche de 0), le neurone a bien appris.

En Annexe 5, 6, 7, et 8 on a les courbes MSE de chaque fonction en fonction du nombre d'itérations. Cela montre que le neurone converge bien.

COMPARAISON DES FONCTIONS D'ACTIVATION

Nous avons comparé trois fonctions d'activation :

- Heaviside : apprend rapidement mais donne des sorties trop brutales (0 ou 1). Il est sensible au bruit.
- Sigmoïde : plus douce, meilleure gestion du bruit.
- ReLU : bon compromis, un peu lent mais plus stable que Heaviside.

Voir en annexe 9 le tableau de taux de réussite entre ReLU, Heaviside , Sigmoide

TESTS ET PREDICTIONS SUR FICHIERS INCONNUS

Une fois le neurone entraîné, nous avons mis à l'épreuve sa capacité de généralisation, en lui donnant un enregistrement inconnu. Ce dernier contient soit des aboiements, soit des miaulements, voire un mélange des deux.
Comme précédemment, nous avons découpé cet enregistrement en plusieurs tranches, calculé son spectre via la FFT, puis extrait les vecteurs correspondants à chaque tranche afin de les présenter au neurone.

Le neurone nous a alors retourné un score pour chaque tranche :

- proche de 0 → interprété comme un « chien »
- proche de 1 → interprété comme un « chat »

Nous pouvons comparer cette sortie à la bonne réponse attendue.

ANALYSE CRITIQUE DES RÉSULTATS

04

TAUX DE RECONNAISSANCE

Grâce à la structure du code, nous avons pu tester trois types de neurones avec différentes fonctions d'activation :

- Heaviside (neurone à seuil)
- Sigmoïde (activation lisse)
- ReLU (activation moderne, rapide)

Chaque neurone a été entraîné avec 20 extraits de miaulements et 20 extraits d'aboiements, en extrayant les vecteurs FFT normalisés. Puis, le neurone a été testé sur un fichier audio long inconnu.

Points à relever :

- Le ReLU a la plupart du temps un bon taux de reconnaissance (80%) avec des sorties entières et gère efficacement les données.
- Le neurone sigmoïde a également de bons résultats (environ 90%) et sur beaucoup plus d'itérations.
- Le neurone Heaviside est très vite entraîné mais les sorties binaires sont source de sensibilité et de variabilité (pour un même modèle entraîné plusieurs fois, le taux peut particulièrement varier), les taux de reconnaissance sont souvent en dessous des autres (autour de 75%).

Tout cela démontre bien que la fonction d'activation a son importance dans la qualité de la classification notamment lorsque l'on applique du bruit sur nos entrées.

AVANTAGES / LIMITES DU MODÈLE

Avantages :

- De bout en bout (acquisition du son, FFT, extraction, neurone), le système est simple et organisé en modules interchangeables.
- Le modèle est efficace sur un petit jeu d'entraînement et le code Java est assez clair pour être réutilisé.
- La sérialisation du neurone rend inutile la longue phase d'apprentissage à chaque lancement du système.
- La mise en œuvre de trois types de neurones différents (le perceptron, le réseau de neurone à une couche, le réseau de neurone à deux couches) montre la modularité du modèle.

Limites :

- La taille des jeux d'entraînements est trop faible (20 extraits) pour garantir une bonne capacité de généralisation.
- Le modèle ne prend pas en compte la temporalité entre les segments (un segment est considéré indépendamment des autres).
- Le perceptron a été rapidement dépassé par la complexité du problème, notamment en présence d'enregistrements parasités ou déformés.
- L'algorithme de normalisation est probablement perfectible malgré sa simplicité.

PERSPECTIVES D'AMÉLIORATION

Pour améliorer le système, plusieurs options sont envisageables:

- Augmenter la taille du jeu de données et collecter davantage de sons différents pour l'entraînement.
- Utiliser un réseau de neurones plus complexe pour pouvoir exploiter la complexité des sons ou leurs variations dans le temps.
- Améliorer le traitement des signaux audios (réduction de bruit, extraction de features plus complexes, ...).
- Appliquer des techniques d'augmentation de données afin de d'augmenter la robustesse du modèle.
- Améliorer la standardisation et la sélection des fréquences utilisées en entrée.
- Ajouter une méthode de calcul du taux de reconnaissance automatique, et si possible compléter avec d'autres métriques (précision, rappel, etc...).

SOLUTION FINALE

1. Chargement des arguments et vérifications

Le programme attend 4 fichiers en entrée :

- Un fichier contenant des miaulements pour l'entrainement
- Un fichier contenant des aboiements pour l'entrainement
- Un fichier audio long à analyser qui est le test
- Le type de neurone à utiliser (R, H ou S)

Il vérifie que tous les arguments sont là, et que le type de neurone est bien reconnu.

2. Sélection du neurone

Le programme choisit automatiquement :

- ReLU, Heavyside ou Sigmoïde, selon la lettre donnée
- Le fichier .txt correspondant au neurone déjà entraîné

3. Chargement ou entraînement du neurone

- S'il trouve un fichier de neurone déjà sauvegardé, il le charge.
- Sinon, il entraîne un nouveau neurone :
  - Il extrait 20 blocs audio de chat et 20 de chien.
  - Il applique la FFT à chaque bloc pour en faire un vecteur de fréquences.
  - Il les normalise (valeurs entre 0 et 1).
  - Il associe 1 au chat, 0 au chien, et entraîne le neurone avec.

4. Analyse d'un fichier long

- Le fichier est découpé en blocs de 1024 échantillons (≈ 0.02 s).
- Chaque bloc est converti en spectre via FFT.
- Le spectre est donné au neurone.
- Le neurone prédit si c'est un chat ou un chien.

5. Affichage des résultats

- Pour chaque bloc, il affiche le résultat (chat ou chien), la sortie du neurone, et la confiance pour les fonctions Sigmoïde et Heaviside.
- Il compte le nombre de chats et de chiens détectés.
- Il affiche aussi le taux de réussite global, c'est-à-dire combien de blocs ont été bien classés (si l'on sait que la première moitié est du chat, la seconde du chien).

Conditions que respectent le code

- Vérifie que les bons fichiers audio sont fournis
- Vérifie que le type de neurone est correct (R, H ou S)
- Recharge le neurone existant si possible
- Sinon, réentraîne le neurone et le sauvegarde
- Applique systématiquement la FFT + normalisation
- Utilise la bonne méthode selon le type de neurone

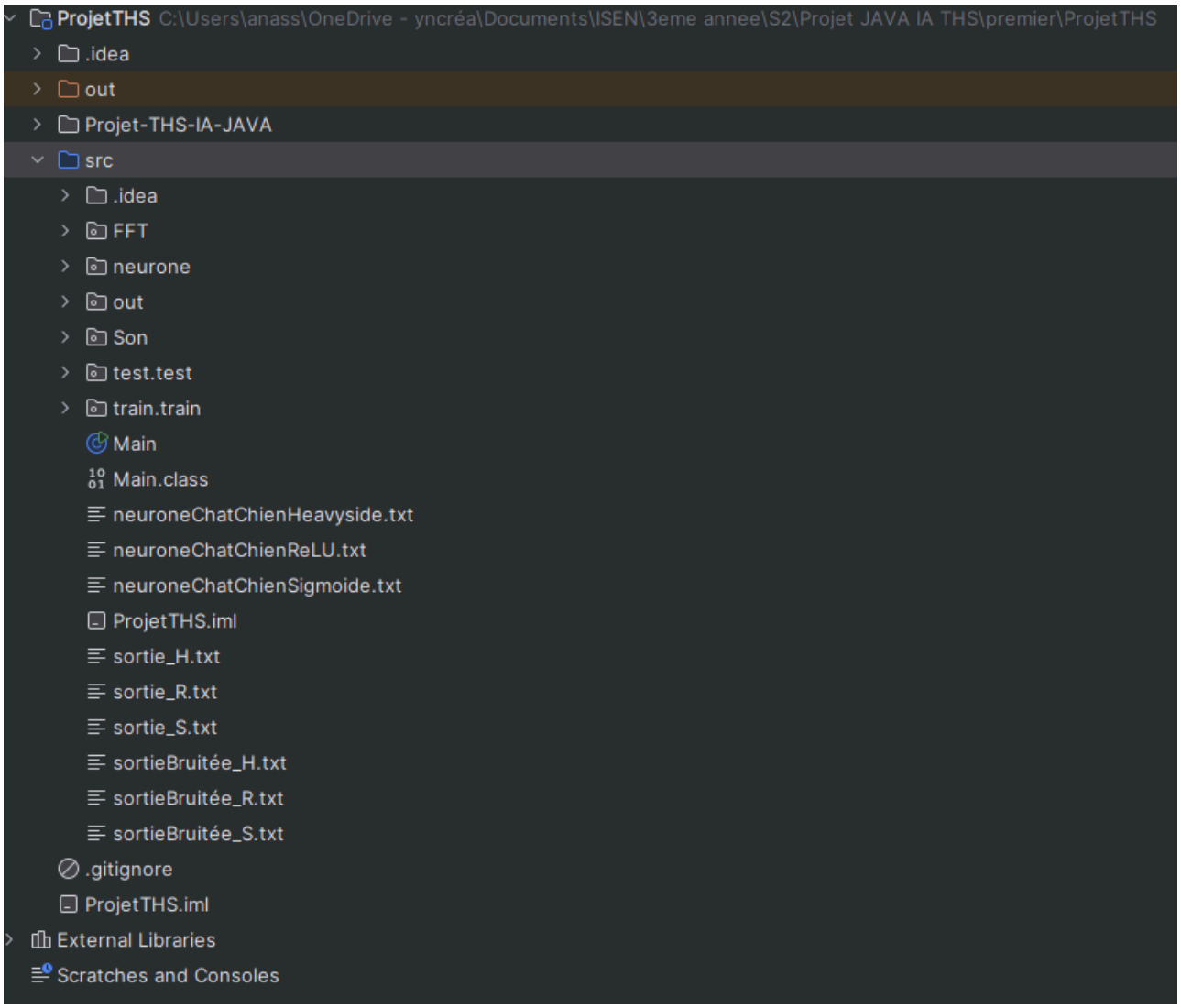
CONCLUSION

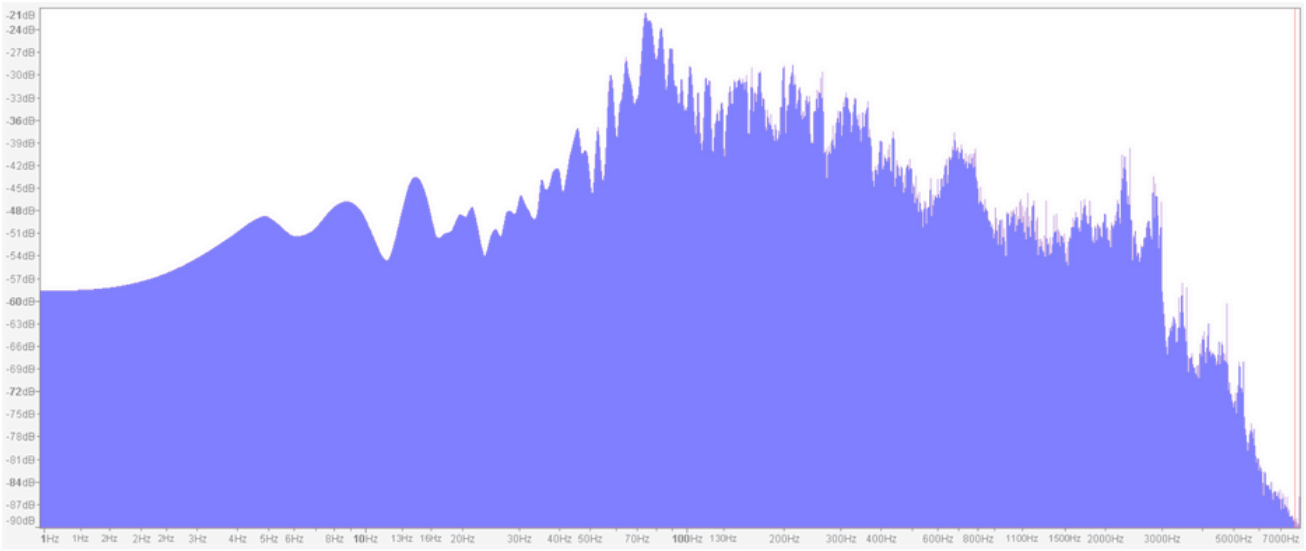
Ce travail a montré qu'il est envisageable de classer des sons assez simples grâce à la FFT et à un perceptron. Les résultats sont encourageants pour un modèle très basique et nombreuses sont les améliorations possibles à l'avenir qui permettrait d'améliorer la précision d'un tel système. Pour y parvenir et améliorer les performances du système, plusieurs pistes peuvent être explorées :

- Augmenter la quantité et la variété des sons présents dans le jeu de données d'entrainement.
- Utiliser un réseau de neurones à couches multiples pour modéliser de manière plus complexe les formes des sons à reconnaitre et la variation de ces derniers dans le temps.
- Améliorer l'étape de pré-traitement du signal audio (fenêtrage, suppression de bruit, extraction des caractéristiques plus complexes…)
- Appliquer des techniques d'augmentation de données afin d'améliorer la robustesse eu égard à des variations de conditions d'acquisition des signaux audio.
- Expérimenter avec l'étape de normalisation ainsi que la sélection des fréquences utilisées comme entrées.

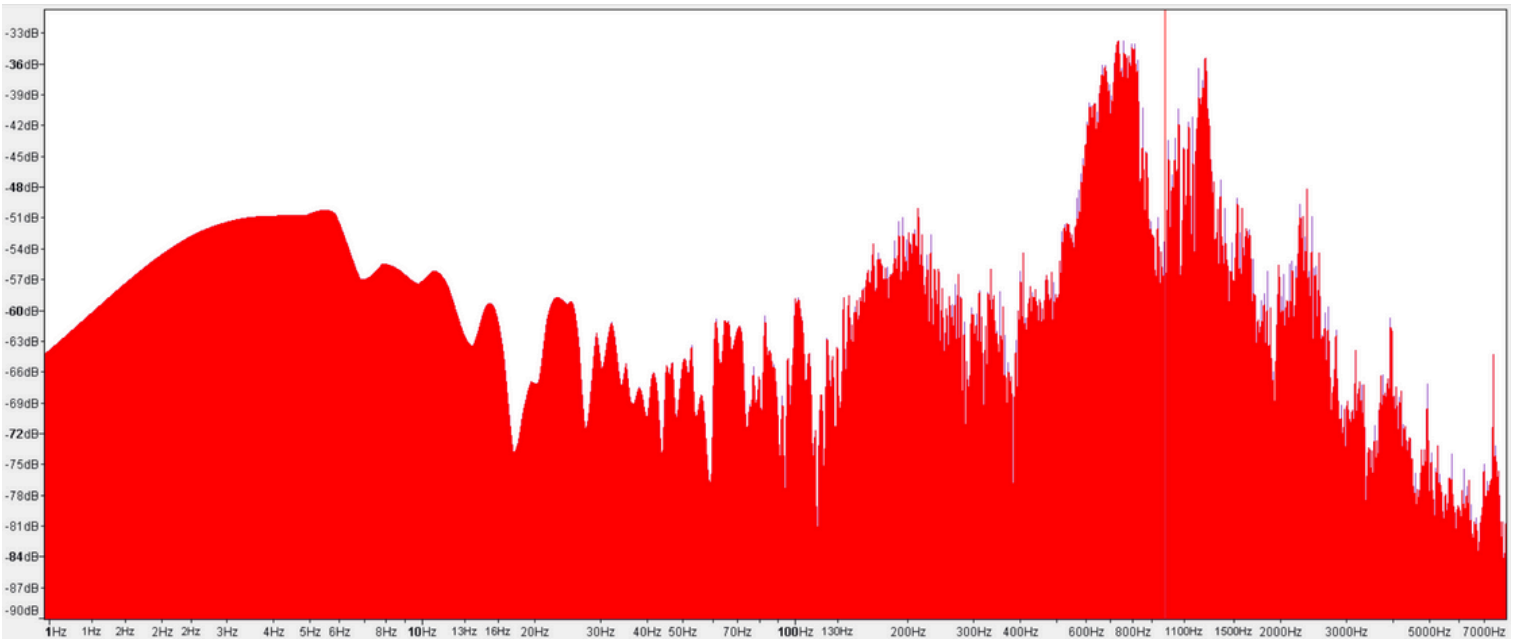
ANNEXES

ARBORESCENCE DU PROJET

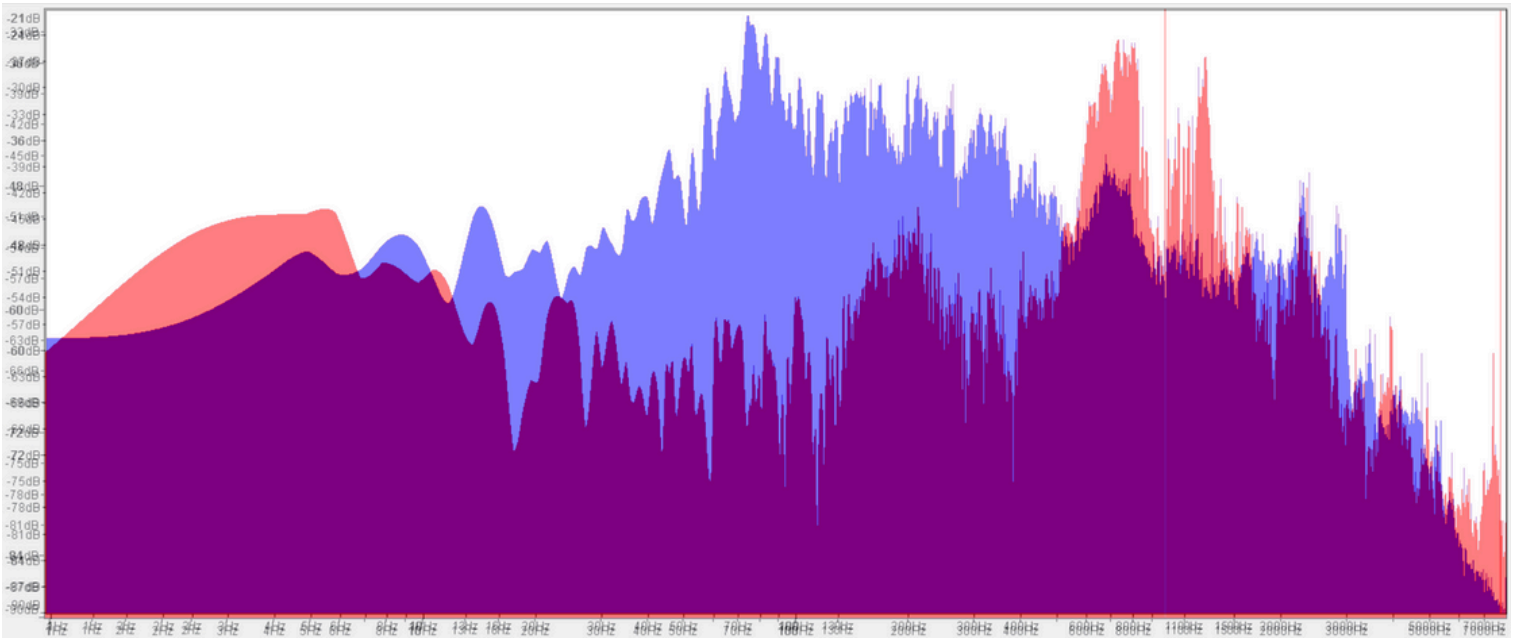




ANNEXE 1 => FFT CHAT



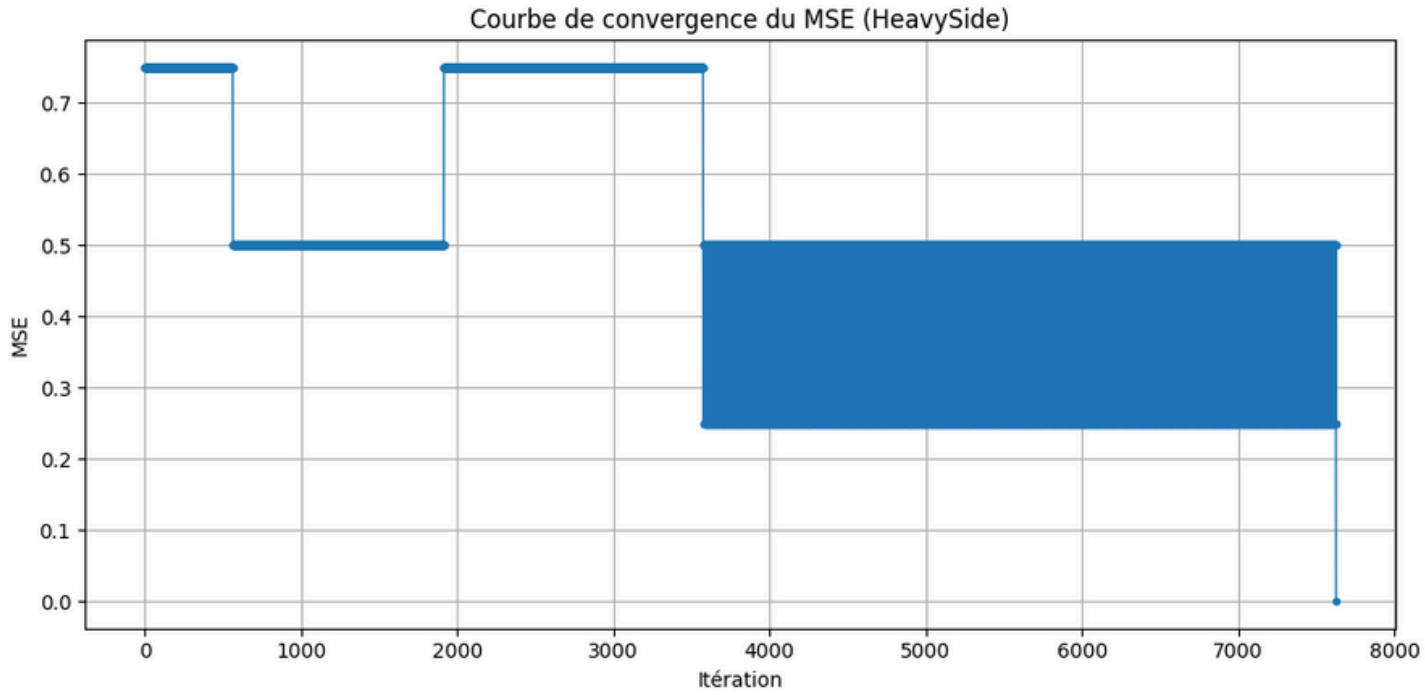
ANNEXE 2 => FFT CHIEN



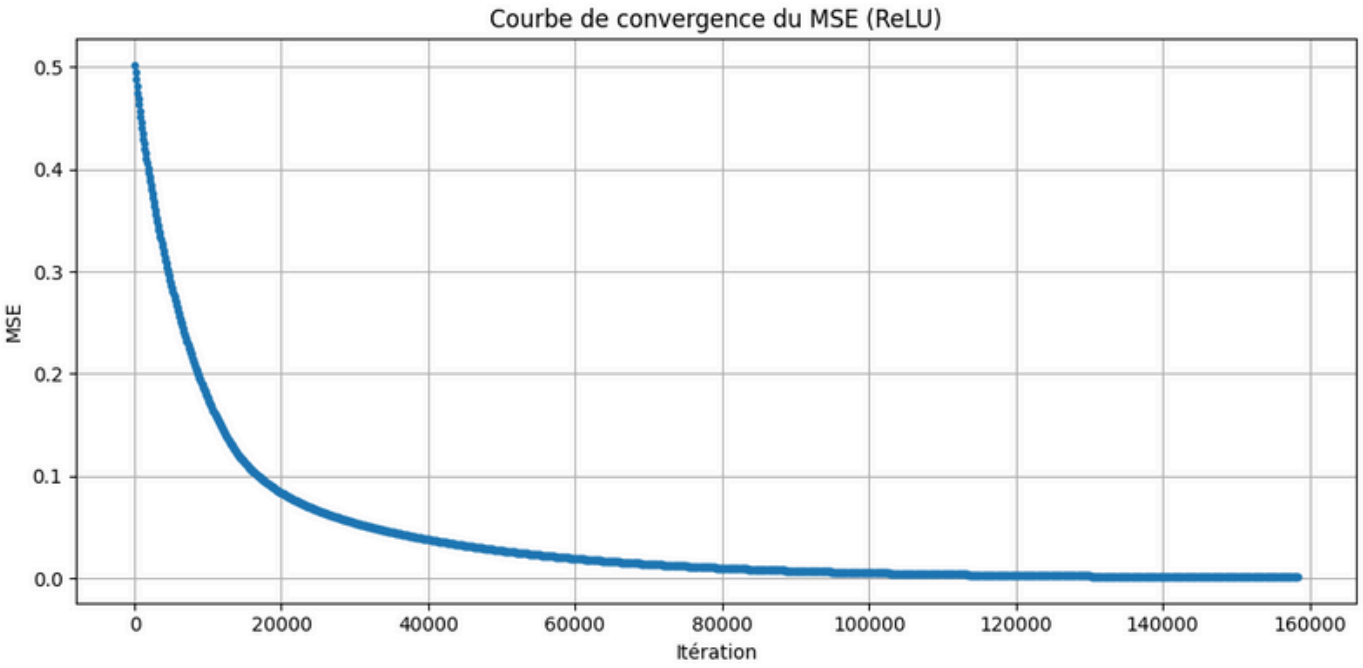
ANNEXE 3 => FFT CHAT, CHIEN

```java
// Fonction d'apprentissage relative à la MSE (Mean Squared Error)
// Elle ajuste les poids (synapses) et le biais du neurone jusqu'à ce que l'erreur soit inférieure à un seuil donné.
public void apprentissage(final float[][] entrees, final float[] resultats, final float MSElimite)
{
    double mse = 0.;     // Initialisation de l'erreur quadratique moyenne
    int iter = 0;        // Compteur d'itérations

    do
    {
        mse = 0.;        // Réinitialisation de l'erreur à chaque itération

        // Parcours de tous les exemples d'apprentissage
        for (int i = 0; i < entrees.length; ++i)
        {
            final float[] entree = entrees[i];   // Entrée courante

            metAJour(entree);                    // Calcul de la sortie actuelle du neurone pour cette entrée
            final float delta = resultats[i] - sortie();  // Calcul de l'erreur (différence entre sortie attendue et réelle)
            mse += delta * delta;                // Accumule le carré de l'erreur pour le calcul final de la MSE

            // Mise à jour de chaque poids synaptique selon la règle delta : w_j ← w_j + η * x_j * δ
            for (int j = 0; j < entree.length; ++j)
                synapses()[j] += entree[j] * eta * delta;

            // Mise à jour du biais selon la règle : b ← b + η * δ
            fixeBiais(biais() + eta * delta);
        }

        mse /= entrees.length;   // Moyenne des erreurs quadratiques (MSE)
        System.out.printf("Itération %d, mse:  %.6f\n", iter, mse);  // Affichage de l'erreur à chaque itération
        iter += 1;

    }
    while (mse > MSElimite);  // Continue tant que l'erreur reste au-dessus du seuil fixé
}
```

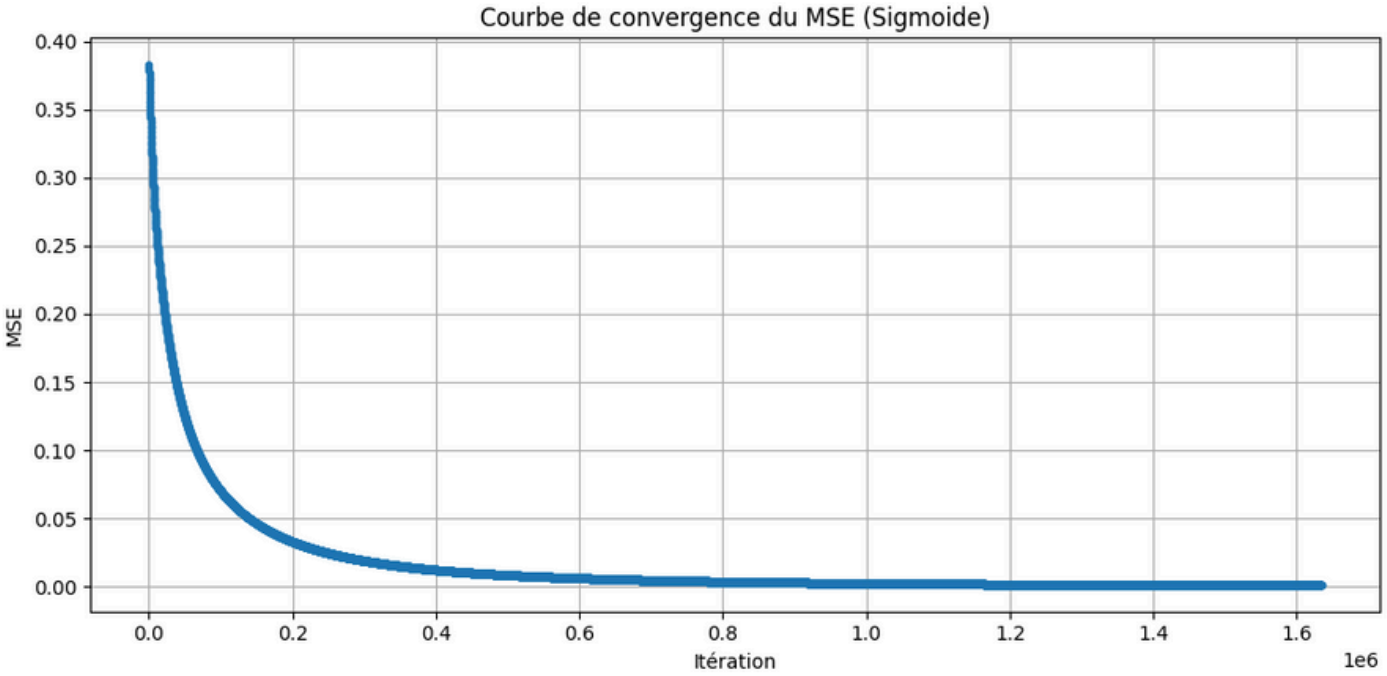
ANNEXE 4 => CODE APPRENTISSAGE



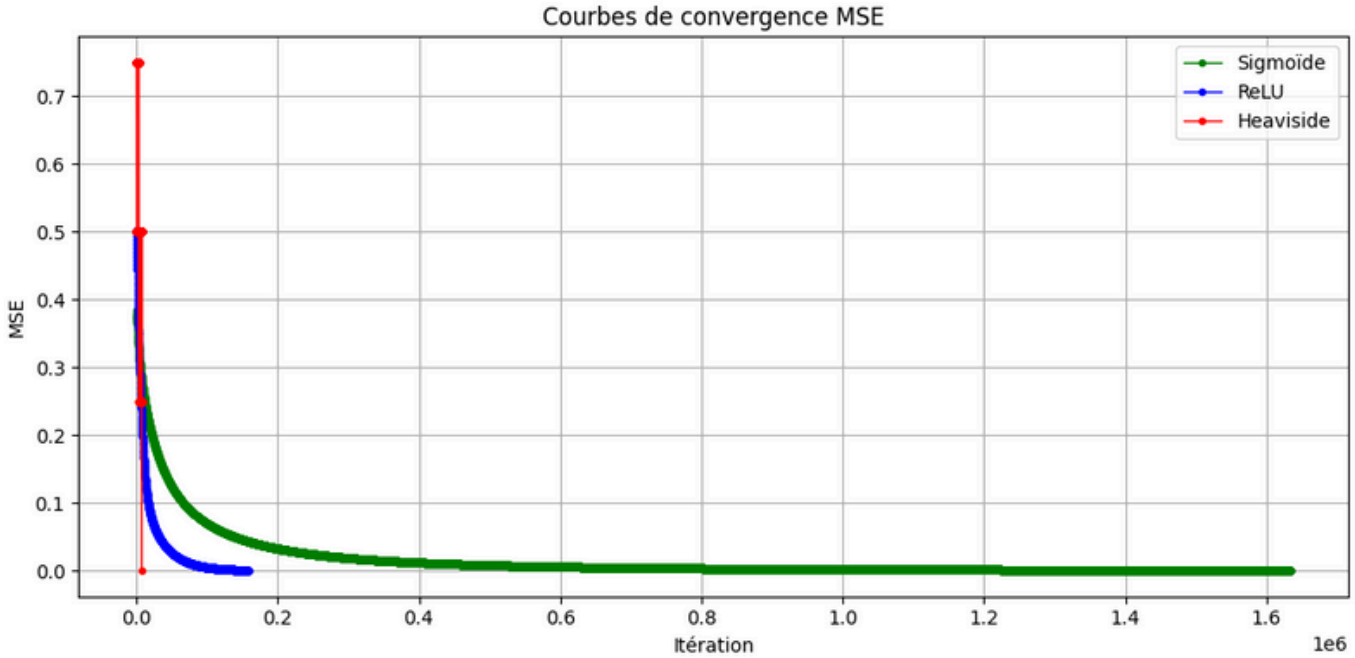
ANNEXE 5 => MSE HEAVYSIDE



ANNEXE 6 => MSE RELU



ANNEXE 7 => MSE SIGMOIDE



ANNEXE 8 => MSE COMPARAISON FONCTION

Heaviside	66%
ReLU	83,30%
Sigmoïde	92,72%

ANNEXE 9 => TABLEAU TAUX DE RÉUSSITES