Réseau de neurones et déconvolution - Application à la simulation d'un amplificateur guitare

Pt.II - Mise en place du dispositif

Jean-Loup Pecquais

2024-11-03

Figure 1: Image générée par IA

Dans l'article précédent, nous avions discuté d'un ensemble de problématiques entourant la création de modèles pour les réseaux de neurones dans le but de simuler des amplificateurs de guitare, ainsi que proposé une piste d'amélioration. Nous la résumerons de la façon suivante :

- On capture l'empreinte de l'amplificateur de guitare en enregistrant la sortie de son haut-parleur à l'aide d'un microphone.
- On procède à la création d'une réponse impulsionnelle du haut-parleur grâce à un amplificateur "neutre"
- On déconvolue les empreintes de l'amplificateur guitare avec la réponse impulsionnelle du haut-parleur.

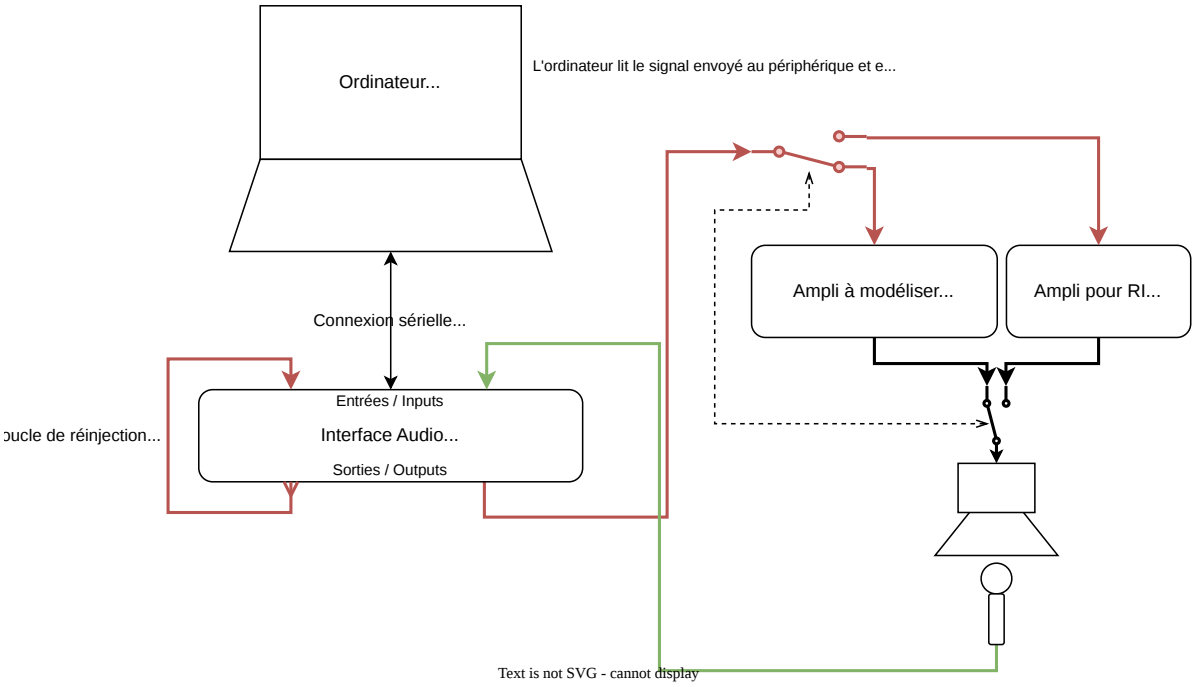

Figure 2: Pour rappel : schéma du dispositif

Voyons maintenant la mise en pratique.

Liste des courses

L'amplificateur que nous souhaitons modéliser est l'amplificateur de puissance d'un Laney LionHeart L20T-410. Le microphone utilisé est un Behringer ECM8000. L'amplificateur utilisé pour réaliser la réponse impulsionnelle est un Drawmer CPA-50.

Pour assurer la simulation d'amplificateur, la solution choisie est le logiciel « AIDA-X ». Plusieurs raisons expliquent ce choix : AIDA-X est une solution open source. Elle offre une extension multi-plateforme et est accessible via la pédale MOD Dwarf. Elle repose sur RTNeural, un framework open source créé par Jatin Chowdhury.

Le protocole de création de modèles pour AIDA-X est détaillé dans leur Google Colab. Le principe est relativement simple. On suit les étapes décrites dans le notebook, on récupère le fichier audio à lire au travers du périphérique à modéliser, on enregistre le résultat, puis on envoie les deux fichiers sur le notebook pour réaliser l'entraînement du réseau de neurones.

Mesure de la réponse impulsionnelle

La réponse impulsionnelle du haut-parleur est réalisée grâce à REAPER. En effet, le plug-in ReaVerb permet de réaliser une déconvolution si le signal utilisé pour la mesure est un "sweep" logarithmique[1]. Pour plus d'information sur la procédure à suivre, voilà un lien vers un excellent article de *Sound on Sound* sur le sujet.

Le signal sort du convertisseur vers un boîtier de réamping[2] pour ensuite attaquer l'entrée de l'amplificateur Dawmer. Le haut-parleur du Laney est directement connecté sur ce dernier.

Une fois tout cela réalisé, on se retrouve avec cette réponse impulsionnelle.

Enregistrement des empreintes

L'amplificateur Laney possède un réglage de présence. Ce type de réglage contrôle en général un filtre passe-bas dans une boucle de rétroaction partant de la sortie de transformateur vers l'entrée de l'amplificateur de puissance.

Il a donc été décidé de créer onze empreintes, une pour chacune des valeurs numériques indiquées par le potentiomètre de présence.

Problématique de la déconvolution des empreintes

Nous voici donc dans le dur du sujet. Reprenons quelques bases de traitement du signal. Lorsque l'on étudie un système linéaire et invariant temporellement[3], on sait que le signal d'entrée et le signal de sortie sont liés à la réponse impulsionnelle du système par l'opération de **convolution**.

$$y(t) = x(t) * h(t)$$

[1] Un sweep est un signal constitué d'un son pur, mais dont la fréquence évolue dans le temps. L'objectif est de couvrir toutes les gammes de fréquences que l'on souhaite examiner. Dans le domaine de la mesure du matériel audio, il est recommandé d'utiliser un balayage logarithmique, où la fréquence augmente ou diminue progressivement selon un schéma logarithmique. Cette approche est avantageuse, car notre sensibilité aux différentes fréquences suit une progression logarithmique.

[2] Un boîtier de réamping adapte le signal de niveau ligne pour une entrée d'amplificateur pour guitare.

[3] Voir wikipedia.

Où x est notre signal d'entré, y notre signal de sortie, et h la réponse impulsionnelle du système.

Nous savons qu'une des propriétés très intéressantes de la transformée de Fourier est de transformer une convolution temporelle en multiplication fréquentielle.

$$Y(f) = X(f) \times H(f)$$

Où Y, X et H sont les transformée de Fourier respective de y, x et h.

> **!** Important
>
> N'oublions pas, $Y(f)$, $X(f)$ et $H(f)$ sont des nombres complexes[4].

Dès lors, si nous connaissons le signal d'entrée, le signal de sortie et que nous cherchons la réponse impulsionnelle, on trouve facilement que :

$$H(f) = \frac{Y(f)}{X(f)}$$

Seulement, l'opération de division est problématique, puisque, si, pour certaines valeurs de f, $X(f)$ vaut zéro, $H(f)$ tendra vers l'infini. Cela se traduira par l'apparition de "sifflantes" dans le signal audio déconvolué.

Une solution classique au problème consiste d'abord à transformer notre expression précédente afin que le dénominateur de la fonction devienne un réel, puis à y ajouter une erreur pour limiter l'influence des valeurs de $X(f)$ trop proches de zéro.

On commence donc par multiplier le dénominateur et le numérateur de notre expression de $H(f)$ par le complexe conjugué de $X(f)$, que nous écrirons $X^*(f)$. On trouve donc :

$$H(f) = \frac{Y(f).X^*(f)}{X^2(f)}$$

On définit maintenant une erreur, soit constante ϵ, soit variable en fonction de la fréquence $\epsilon(f)$. Il en découle alors :

$$H(f) = \frac{Y(f).X^*(f)}{X^2(f) + \epsilon^2(f)}$$

Il n'existe pas, à ma connaissance, de méthode classique pour déterminer ϵ. Il est d'usage d'ajouter l'erreur la plus faible possible, afin de limiter l'approximation de l'opération de

[4] Voir wikipedia

déconvolution. Dans la même logique, chercher une erreur variable en fonction de la fréquence est en général la meilleure approche.

> **Astuce**
>
> La proposition d'une méthode pour approcher une valeur d'ϵ fera peut-être l'objet d'un article. Un jour.

On appelle cette méthode de déconvolution la déconvolution de Weiner.

Implémentation de la déconvolution

À ma connaissance, il n'existe pas d'outil de déconvolution gratuit n'imposant pas de contrainte sur le signal utilisé pour la mesure. Le plus simple fut donc d'implémenter l'algorithme de déconvolution en Python, en utilisant les librairies Numpy et SciPy.

Le choix a été fait d'implémenter la déconvolution comme la convolution par la réponse impulsionnelle de l'inverse du spectre du signal d'entrée. Le formalisme mathématique peut ici nous éclairer. Nous avons écrit plus haut :

$$H(f) = \frac{Y(f)}{X(f)}$$

Nous pouvons tout aussi bien écrire :

$$H(f) = Y(f) \times \frac{1}{X(f)}$$

Soit, par transformée de Fourier inverse :

$$h(t) = y(t) * x_{inv}(t)$$

Où $TF(x_{inv}(t)) = \frac{1}{X(f)}$

Rappelons-nous tout de même que nous souhaitons limiter l'apparition de sifflante par l'ajout d'une erreur. On pause alors la relation suivante :

$$TF(x_{inv}(t)) = \frac{X^*(f)}{X^2(f) + \epsilon^2(f)}$$

Voici une proposition d'implémentation en Python:

```
def invert_filter(impulse: np.ndarray, epsilon: np.ndarray = 0):
    Impulse = np.fft.fft(impulse)
    Kernel = np.conj(Impulse) / (Impulse*np.conj(Impulse)+np.power(epsilon,2))
    return np.real(np.fft.ifft(Kernel))
```

Cette implémentation présuppose que le paramètre `epsilon` de la fonction est soit un nombre, soit une liste de même taille que le paramètre `impulse`. Le code suivant propose un exemple de l'utilisation de cette fonction :

```
import numpy as np
from scipy import io, signal

# Fonction de génération de "l'inverse" de la réponse impulsionnelle
def invert_filter(impulse: np.ndarray, epsilon: np.ndarray = 0):
    Impulse = np.fft.fft(impulse)
    Kernel = np.conj(Impulse) / (Impulse*np.conj(Impulse)+np.power(epsilon,2))
    return np.real(np.fft.ifft(Kernel))

# Conversion décibel vers linéaire
def db2a(x):
    return np.power(10,x/20)

# Définition de la fréquence d'échantillonnage de travail
fs = 48000

# On charge l'IR et l'empreinte
impulse_sr, impulse = io.wavfile.read('impulse_response.wav')
model_sr, model = io.wavfile.read('model.wav')

# On définie le niveau de bruit que l'on va rajouter pour la déconvolution. Ici, l'erreur est
noise_floor = -100 # dB

# la variable kernel stocke "l'inverse" de notre réponse impulsionnelle
kernel = invert_filter(impulse,db2a(noise_floor))

# La fonction lfilter permet de réaliser l'opération de convolution.
output = signal.lfilter(kernel,a=1,x=model)

# Cette normalisation est recommandée par AIDA-X
# https://mod.audio/modeling-amps-and-pedals-for-the-aida-x-plugin-best-practices/
output /= (np.max(output)*db2a(6))
```

```
# On écrit la variable output dans un fichier
io.wavfile.write('model_deconvolved.wav',fs,output)
```

Résultats

Une fois le script exécuté, on obtient notre fichier cible pour le réseau de neurones, où l'empreinte sonore du haut-parleur a été retirée.

En guise de comparaison, voici le même signal audio, avant et après déconvolution :

Dans le prochain article, nous regarderons comment implémenter ce modèle sur une carte Bela, puis nous prendrons le temps d'écouter le résultat.