I. Introduction et problématique

Aujourd'hui, on dispose de moyens assez efficaces pour transformer automatiquement un texte prononcé dans un fichier audio en sa transcription écrite (par exemple, l'outil de sous-titrage automatique de Youtube). Le problème avec ces outils est qu'ils nécessitent presque toujours de savoir en quelle langue le texte est prononcé pour pouvoir le transcrire efficacement.
L'objectif de ce projet d'initiation à la recherche est de parvenir à détecter automatiquement la langue parlée dans un fichier audio. Cette détection automatique pourrait permettre de directement transcrire le discours prononcé dans le fichier, en utilisant la "bibliothèque" vocale correspondant à la langue parlée.
Pour ce faire, on utilisera des méthodes d'apprentissage automatique (machine learning) basées sur des réseaux de neurones profonds.
Dans un premier temps, le but sera de construire un réseau de neurones capable de déterminer, pour un fichier audio donné en entrée, s'il s'agit d'un discours prononcé en allemand, anglais, arabe ou français.
Ensuite, selon les performances du réseau précédent, nous tenterons de créer un système capable de détecter la langue d'un contenu audio de manière dynamique. Ainsi, nous pourrions repérer automatiquement, par exemple, une citation prononcée en anglais dans un contenu globalement allemand, pour pouvoir ainsi faciliter la transcription automatique d'un tel texte.

II. Présentation générale

A. Les réseaux de neurones artificiels

*** Petit schéma réseau de neurones ***

Les réseaux de neurones sont des modèles mathématiques qui sont souvent utilisés dans le domaine de l'apprentissage automatique, en tant qu'outils permettant de faire de la reconnaissance de formes, sons, etc. Grâce à des « couches » successives (le réseau) d'unités de calcul (les neurones) qui sont « excités » ou non selon leur entrée (ce qui donne un résultat binaire) et qui « s'autorégulent » pour s'approcher du résultat attendu, ces outils, si on leur donne un ensemble de données d'apprentissage suffisamment vaste, peuvent détecter des caractéristiques (comme la langue d'un discours) de

manière assez précise.

L'utilisation d'un réseau de neurones consiste en deux phases : tout d'abord l'apprentissage, où l'on va fournir au réseau un ensemble de données « étiquetées » (dans notre cas, des discours marqués comme étant dans une langue en particulier) de manière à ce qu'il puisse déterminer si il a réussi à donner la bonne réponse ou non, et se réguler en conséquence. Une fois que le réseau obtient une marge d'erreur suffisamment basse sur cet ensemble d'apprentissage, on peut l'utiliser pour faire de la détection sur d'autres ensembles de données inconnues, qui ne sont donc pas étiquetées.

On dit qu'un réseau de neurones est "profond" si il est constitué de plusieurs couches de neurones successives : ces réseaux sont souvent plus efficaces sur des problèmes de reconnaissance complexes car les multiples couches leur permettent "d'affiner" successivement ce qui est reconnu (par exemple, pour de la reconnaissance d'images, une première couche pourrait distinguer plusieurs zones de l'image, une seconde les contours des formes qui apparaissent dans ces zones, etc.)

B. Notre travail

L'objectif de ce projet de recherche n'est pas de construire un réseau de neurones par nous-mêmes (nous n'en aurions pas le temps). Pour cette raison, nous allons utiliser une librairie python, appelée Keras, qui permet de créer des réseaux de neurones standard de manière très simple et rapide. Notre but sera donc de comprendre comment fonctionne cette librairie, puis de l'utiliser pour construire un (ou plusieurs) réseau qui permettrait de répondre à notre problématique (qui peut donc se résumer par : "Est-il possible d'utiliser un réseau de neurones pour détecter automatiquement la langue d'un contenu audio ?").

Néanmoins, avant de pouvoir utiliser un réseau de neurones, il est nécessaire de traiter correctement les données dont nous disposons en entrée. Ce prétraitement sera une part importante de notre travail.

Nous avons à disposition un corpus de données audio provenant de sources diverses (émissions de radio, conférences...). Comme tout enregistrement audio, ces fichiers contiennent un certain nombre de passages qui ne sont pas de la parole à proprement dit (musique, applaudissements, silences...), et si ces passages sont trop présents ou trop réguliers, cela risque de biaiser les résultats de notre réseau[1]. Dans un premier temps, nous traiterons donc les fichiers audio afin de supprimer de tels

1 : Par exemple, si des applaudissements ne sont présents que dans les fichiers étiquetés comme étant de l'anglais, le réseau risque de considérer tout applaudissement qu'il pourrait entendre dans le futur comme étant de l'anglais. Cela fausserait à la fois la reconnaissance d'un passage non-anglais dans lequel se trouvent des applaudissements, mais aussi celle des passages en anglais sans applaudissements, car le réseau pourrait considérer que leur absence indique que le morceau n'est pas de l'anglais.

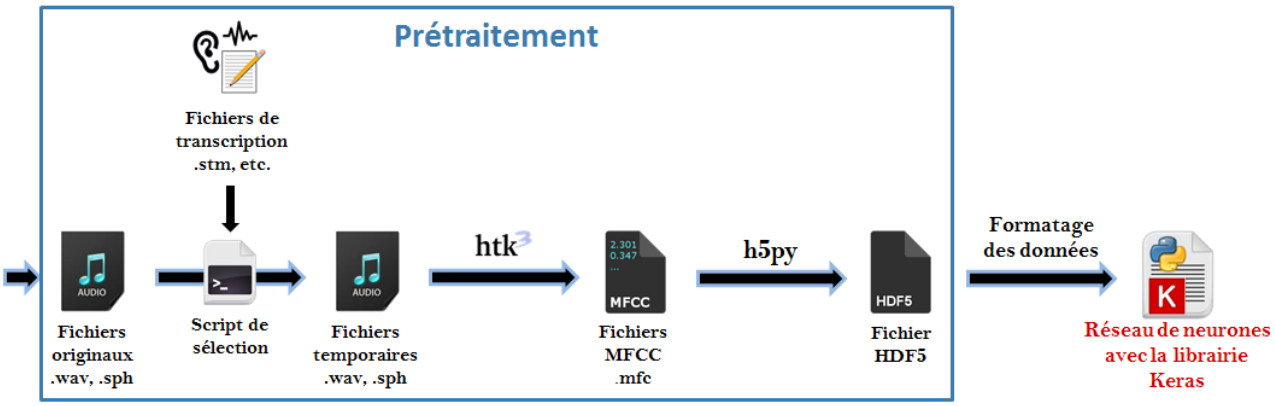
passages.

Se pose ensuite la question de la représentation des données. En effet, un fichier audio est la plupart du temps représenté par un "simple" signal, tandis que le couche d'entrée du réseau attend un certain nombre de valeurs réelles (une par neurone). Nous devons donc transformer les fichiers audio en ensembles de valeurs réelles, à l'aide de transformées de Fourier. Nous utiliserons pour cela un logiciel appelé HTK, qui nous permettra de transformer les signaux audio en ensembles de coefficients acoustiques de manière très simple.

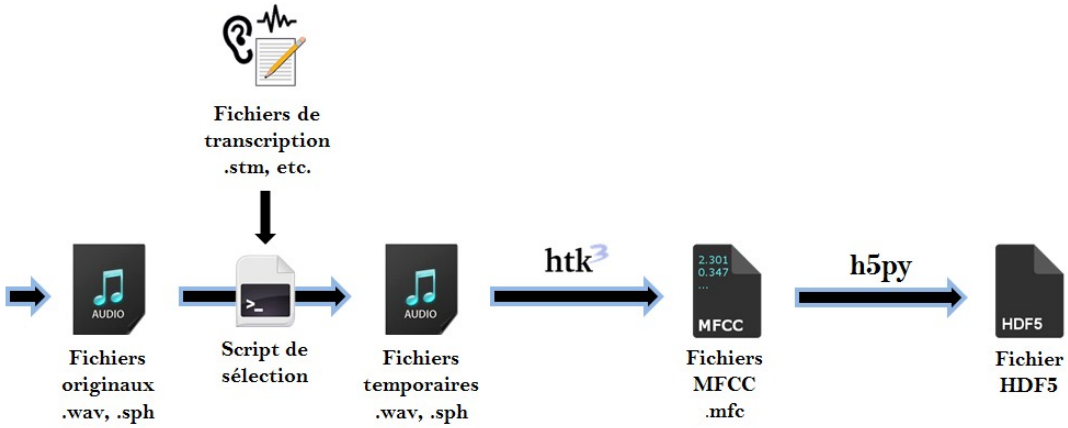
Pour stocker ces ensembles de données, nous utiliserons le format de fichier HDF5, qui permet de classer des groupes de données de manière pratique dans un seul fichier, et qui offre la possibilité d'être lu directement sur le disque dur (sans être chargé en mémoire centrale), ce qui pourrait nous être utile par la suite.

Quand nous aurons crée ce fichier contenant nos données d'apprentissage, le prétraitement sera terminé. Cependant, avant de pouvoir entrer toutes ces valeurs dans le réseau de neurones, il nous sera encore nécessaire de les formater correctement; en effet, nous ne pouvons pas simplement "envoyer" l'ensemble de données d'apprentissage directement dans le réseau : selon son type, il faudra les séparer en plusieurs sous-ensembles (qui pourront même parfois contenir des données de base dupliquées) de taille fixe, et ne pas oublier "d'étiqueter" ces données (afin d'indiquer au réseau ce qu'il est en train d'apprendre).

Enfin, une fois que le format des données sera correct, il nous restera à créer le réseau proprement dit en quelques lignes de code, lui faire apprendre toutes l'ensemble d'apprentissage, puis réaliser des tests sur des données nouvelles et affiner ses paramètres pour diminuer son taux d'erreur.



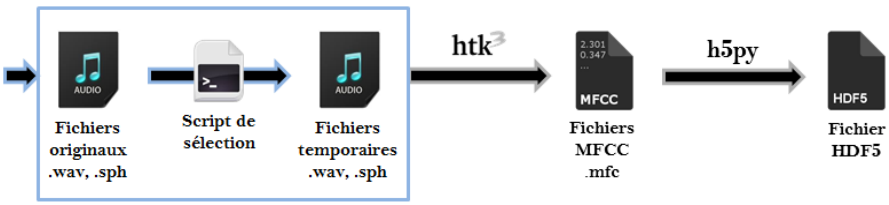
III. Pré-traitement



L'objectif du prétraitement est donc de transformer nos données d'entrée (des ensembles de fichiers audio) en un ensemble de données utilisables par le réseau de neurones (des valeurs réelles), stockées sur le disque dur, pour ne pas avoir à refaire le prétraitement à chaque apprentissage du réseau.

Au départ, nous sommes partis d'un corpus de fichiers de 4 langues différentes (environ une durée totale d'une heure pour chaque) : Arabe, Allemand, Anglais et Français. Les corpus arabe et français sont des extraits d'émissions de radio, le corpus anglais des extraits de conférences, et le corpus allemand des phrases extraites d'un corpus linguistique. En plus des fichiers audio, nous disposons également des fichiers de transcriptions correspondants (sous divers formats) qui nous sont utiles pour le prétraitement.

A. Sélection des données utilisables



Certains des fichiers audio utilisés peuvent poser problème durant la phase d'apprentissage. En effet, certains commencent ou finissent avec un passage musical ou un silence qui représente une partie non négligeable du fichier. Le problème étant que si un silence ou un passage musical est présent au début ou à la fin dans tous les fichiers audio anglais par exemple, le réseau de neurones risque d'apprendre ce silence ou cette musique comme étant de l'anglais et ainsi de biaiser les résultats sur de futures analyses.
De tels problèmes sont présents sur différents fichiers du corpus dont nous disposons :

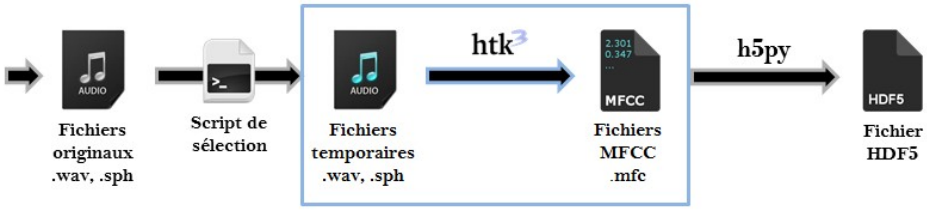
· les extraits en anglais[2] commencent tous par un extrait musical suivi d'applaudissements ;

· les extraits en allemand commencent et se terminent par un silence non négligeable.

Il est donc nécessaire d'enlever ces passages indésirables de manière automatisée (les retirer à la main aurait demandé beaucoup trop de temps étant donné le grand nombre d'extraits, et il aurait été nécessaire de le refaire pour chaque futur corpus). Pour cela, nous utilisons les transcriptions des extraits audio, qui contiennent les temps précis de début et de fin des phrases prononcées.

Nous avons donc écrit un script en Python afin d'extraire les temps de passage de « vraie parole » à l'aide des fichiers de transcription. Pour cela, nous coupons le reste de l'audio des fichiers en utilisant le programme Sox (une boîte à outils permettant de faire diverses opérations sur des fichiers audio).

B. Transformation en coefficients acoustiques



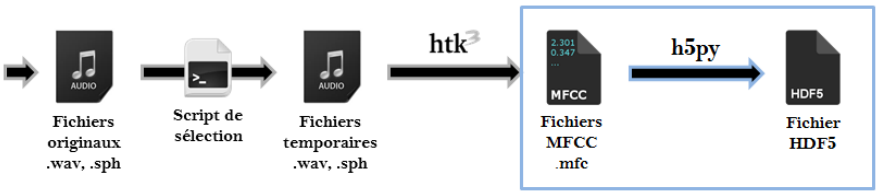
L'étape suivante avant de pouvoir construire un réseau de neurones et de débuter la phase d'apprentissage avec nos données est de traiter les fichiers audio pour les transformer en un format

2 : venant de conférences TEDx

de données utilisable, c'est-à-dire dans notre cas sous la forme de coefficients acoustiques.

Pour effectuer cette conversion, nous utilisons HTK, un logiciel multiplateforme développé par le département d'ingénierie de l'Université de Cambridge, qui est une sorte de « boîte à outils » permettant de réaliser toutes sortes de manipulations utiles à la reconnaissance de la parole. Il permet notamment d'effectuer de manière très simple la conversion de fichiers audio en coefficients acoustiques correspondants : les fichiers MFCC (Mel-frequency cepstral coefficients).
*** Précisions sur les mfcc et le format des vecteurs acoustiques ***

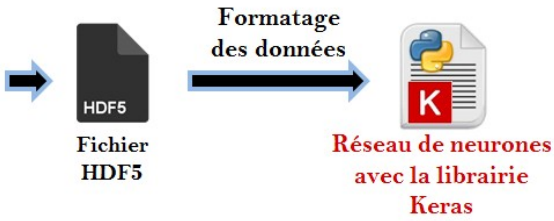
C. Stockage des données



Afin de ne pas répéter la phase de prétraitement à chaque apprentissage du réseau de neurones, il est primordial de stocker les données pré-traitées. Pour cela, nous utilisons le format de données HDF5 qui nous permet de hiérarchiser et d'organiser facilement nos données une fois le prétraitement terminé.
Ce format possède également un autre avantage qui nous est très utile : il permet si nécessaire de transmettre des données en les chargeant directement depuis le disque dur.
En effet, pour éviter un maximum les régularités dans les fichiers de données qui pourraient biaiser le processus d'apprentissage, Keras va mélanger les données qu'il reçoit en entrée pour les traiter dans un ordre aléatoire. Pour ce faire, il est nécessaire de lui fournir l'intégralité des données en une seule fois. De ce fait, la quantité de données générée par la transformation des fichiers audio en coefficients acoustiques peut être trop importante pour pouvoir charger les données directement en mémoire vive selon la taille de départ du corpus. Le format de données HDF5 nous permet donc de parer à ce problème éventuel en chargeant les données directement depuis le disque dur sans surcharger la mémoire vive, même si la vitesse de traitement s'en trouve diminuée.

Afin de manipuler des fichiers HDF5, nous utilisons la librairie python h5py.
*** Précisions sur les deux façons différentes de stocker les données, par fichiers / par phrases ***

IV. Réseaux de neurones



Maintenant que nous disposons des données en bonne forme dans notre fichier HDF5, il est temps de les transmettre au réseau de neurones et de commencer l'apprentissage. Pour cela, il nous faudra formater les données sous la forme attendue par le réseau (qui pourra changer selon le type de réseau construit) et trouver les bons paramètres pour le réseau qui nous permettront d'obtenir de meilleurs taux de reconnaissance.

A. Keras

Keras est une librairie open-source Python utilisée pour construire des réseaux de neurones artificiels, développée par François Chollet dans le cadre du projet de recherche ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). L'avantage de cette librairie est qu'elle est très haut niveau, elle permet donc de créer des réseaux de neurones très rapidement et très simplement. En effet, cette dernière n'est en quelque sorte qu'une "interface de surcouche" qui se place au-dessus de librairies de réseaux de neurones plus bas niveau comme Theano, qui est la sous-couche que nous utiliserons avec Keras.

Il est possible de créer un réseau de neurones (parfois complexe) avec Keras en littéralement quelques lignes :

*** Bout de code commenté de création de réseau simple ***

De plus, Keras est très pratique à utiliser en python (les réseaux crées fonctionnant avec des tableaux numpy classiques[3]) et dispose de beaucoup d'options, qui permettent par exemple d'interrompre l'apprentissage dès qu'on le juge nécessaire, de sauvegarder un modèle de réseau,

3 : ***description rapide de numpy***

d'optimiser la lecture depuis un fichier HDF5...

Dans notre cas, nous utiliserons Keras pour créer deux types de réseaux différents : Un perceptron multi-couches et un réseau LSTM.

B. Mécanismes d'apprentissage

Pour faire l'apprentissage proprement dit, le réseau va procéder de la manière suivante :

- Prendre une certaine quantité (fixe) X de données parmi l'ensemble d'apprentissage qu'il a reçu, de manière aléatoire,
- Passer ces données X à travers les différentes couches du réseau,
- Obtenir le résultat de la prédiction à la couche de sortie, puis le comparer à l'étiquette de X et réajuster les différentes couches par rapport à ce résultat,
- Répéter jusqu'à avoir traité toutes les données de l'ensemble d'apprentissage.

*** Eventuellement petit schéma de description d'une epoch d'apprentissage ***

Ce traitement de toutes les données du corpus d'apprentissage s'appelle une epoch. On effectue généralement plusieurs epochs pour un apprentissage (pour nous, environ une centaine), les performances du réseau étant censées s'améliorer au fil des epochs.

Pour améliorer les capacités d'apprentissage de notre réseau, nous utilisons une méthode classique d'apprentissage automatique qui consiste à séparer nos données d'apprentissage en 2 corpus distincts :

- un corpus d'apprentissage, qu'on va envoyer en entrée du réseau de neurones et qui sera directement "appris",
- et un corpus de validation, sur lequel le réseau va tester ses capacités après chaque epoch, et utiliser les résultats de ce test pour améliorer l'apprentissage sur le corpus de base.

Finalement, si l'apprentissage se déroule conformément à ce qu'on attendait, on utilise le corpus de test pour évaluer les performances de notre modèle sur des données qu'il n'a jamais vues.

C. Perceptron multi-couches

Le premier modèle que nous avons utilisé est un perceptron multi-couches : c'est un des modèles les plus simples à mettre en œuvre mais qui peut donner de très bons résultats.

1. Présentation du modèle

Le perceptron est historiquement un des tout premiers modèles d'apprentissage supervisé.
*** Présentation un peu plus technique du perceptron ***

2. Formatage des données pour ce réseau

Pour ce type de réseau, nous avons utilisé notre premier fichier de données, c'est à dire celui où les données sont classées par fichiers complets.
En entrée du réseau, nous ne pouvons pas simplement mettre un seul vecteur acoustique à la fois : en effet, cela n'a pas vraiment de sens d'essayer d'en prédire la langue puisqu'il correspond à une durée de parole très courte, de quelques millisecondes seulement. Pour cette raison, nous passons en entrée un ensemble de ces vecteurs, que l'on appelle une trame, et dont la durée de parole combinée correspondant aux vecteurs qui la compose fait plutôt quelques dizaines de millisecondes, ce qui peut permettre de plutôt travailler sur des syllabes ou des mots courts. Le nombre de vecteurs composant cette trame est donc un des paramètres de notre réseau.

De plus, afin d'être sûrs de repérer tous les "traits" de caractère possibles d'un langage, nous utilisons une fenêtre glissante : plutôt que de séparer un fichier d'entrée en le découpant en X trames successives, nous "décalons" chaque trame successive d'un certain nombre de vecteurs; ainsi, nous obtenons un ensemble de trames où les n premiers vecteurs de l'une sont les n derniers vecteurs de la précédente. Ce décalage est donc aussi un paramètre de notre réseau.

*** Petit schéma avec le découpage en trames d'un fichier ***

3. Format du réseau

Au final, pour ce perceptron multi-couche, nous utilisons deux couches de taille (nombre de valeurs

dans un vecteur acoustique * nombre de vecteurs dans une trame) dont la couche d'entrée, pour avoir effectivement un neurone par valeur réelle, et une couche de sortie de taille 4, puisque nous avons 4 langages possibles différents.
Cette couche de sortie correspond à une fonction softmax : c'est à dire que, pour chaque entrée, le réseau générera 4 probabilités dont la somme est égale à 1 : la probabilité que l'entrée soit du français, la probabilité que l'entrée soit de l'allemand... et donnera en sortie la langue dont la probabilité est la plus élevée.

*** Code commenté de la création du perceptron ***

4. Résultats

Une fois la phase de prétraitement et la mise en place du perceptron avec Keras terminées, nous avons pu effectuer un premier apprentissage qui s'est révélé infructueux. En effet, le réseau apprenait correctement sur le corpus d'apprentissage et s'améliorait linéairement au fur et à mesure mais ne s'améliorait pas sur le corpus de validation et stagnait aux alentours de 50% de précision, ce qui semblait indiquer que le réseau n'apprenait rien d'assez général qui puisse l'aider à s'améliorer sur le corpus de validation.

Nous avons ensuite effectuer d'autres apprentissages en faisant varier les différents paramètres (taille des trames d'entrée, décalage), notamment en augmentant la taille des trames d'entrée de manière significative sur certains essais en partant de la supposition que la taille des trames était peut-être insuffisante pour apporter de l'information pertinente au réseau.
Nous avons également essayé de normaliser les données en mettant la variance de chaque composante des vecteurs acoustiques à 1 pour les corpus d'apprentissage et de validation).
Tous les essais sur le perceptron ont cependant conduit au même résultat que précédemment : aucune amélioration sur le corpus de validation pendant la phase d'apprentissage.

C. LSTM

Les différents essais avec le perceptron n'ayant donné aucun résultat probant, nous avons ensuite utilisé un autre type de réseau de neurones, les réseaux LSTM (Long Short Term Memory), qui ont comme particularité d'être des réseaux récurrents.

1. Présentation du modèle

Le principe d'un réseau récurrent est de se "souvenir" des résultats de l'apprentissage de données actuelles pour essayer d'améliorer sa performance sur les données à venir :autrement dit, en donnant de l'importance aux trames précédentes lors de l'évaluation de la trame actuelle, on utilise un contexte qui peut aider à prendre une décision.

2. Formatage des données pour ce réseau

Utiliser des phrases de taille minimale fixe découpées en ensembles de séquences //A préciser/compléter

3. Format du réseau

4. Résultats

V. Conclusion