INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport des travaux effectués sur la nasalisation

Projet scientifique

24 Mai 2020

Raphaël BOURGEOIS
Clément CHIDEKH

Rapport des travaux effectués sur la nasalisation Projet scientifique

24 Mai 2020

Raphaël BOURGEOIS
Clément CHIDEKH

Table des matières

Introduction

Au niveau de l'articulation, la nasalisation correspond à la production d'un son lorsque le voile du palais est abaissé par le relâchement du *levator palatini*, principal muscle responsable de l'élévation du voile du palais, de telle sorte que l'air puisse s'échapper par le nez durant la production du son. Il se produit alors une résonance nasale, qui auditivement se traduit par la production d'un son [n] pour les voyelles nasales. Par exemple, le son « a » devient « an », le « o » devient alors un « on ». On dit que les voyelles passent du timbre oral au timbre nasal.

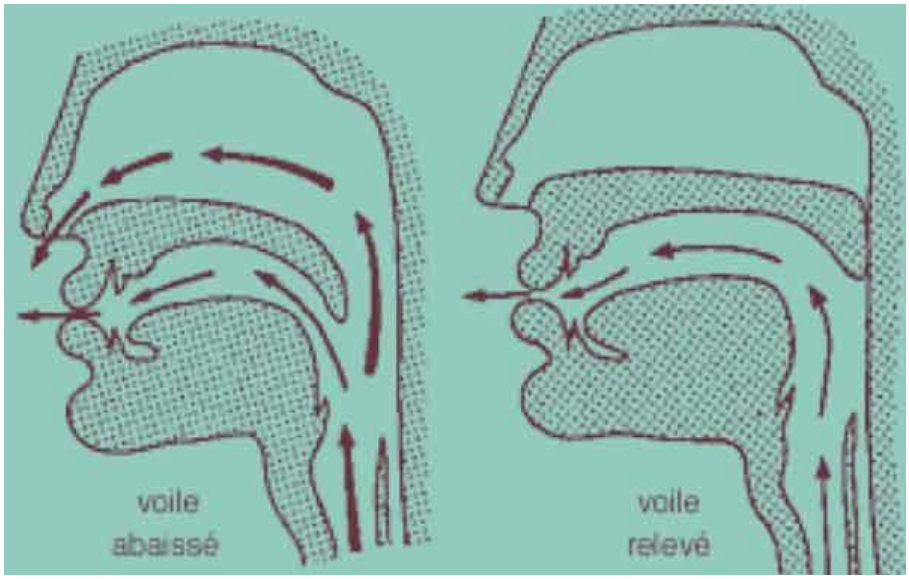

Figure 1 – Schéma montrant relâchement du levator palatini

La figure 1 nous montre le trajet de l'air expiré lors de la production d'un son nasalisé (à gauche) et d'un son vocal (à droite). Dans l'alphabet phonétique international, la nasalisation est indiquée par la présence d'un tilde au-dessus du symbole du son à nasaliser. Les voyelles nasales sont présentes à hauteur de 5 dans la langue française avec [Ã], [Õ], [Ũ], [Ĩ] même si la différenciation entre les deux dernières se fait de plus en plus menue: par exemple, on différencie très peu les mots "brin" et "brun" (Nasalisation, s.d.). Aujourd'hui, parmi les 700 langues répertoriées par Ruhlen, 1975, 150 ont des voyelles nasales (Vaissière, 1995).

1. Les caractéristiques du spectre d'une voyelle nasalisée

Dans cette partie, on s'intéressera exclusivement à la voyelle nasale [Õ] qui s'entend "on" et à son équivalent vocal [O]. Afin de déterminer les caractéristiques du [Õ] et ses différences avec le [O] ne nous appuierons sur les enregistrements de ces syllabes dans le contexte de la phrase "Bonjour Monsieur Tralipo". La figure 2 montre les spectres obtenus à l'aide du logiciel Audacity (Audacity).

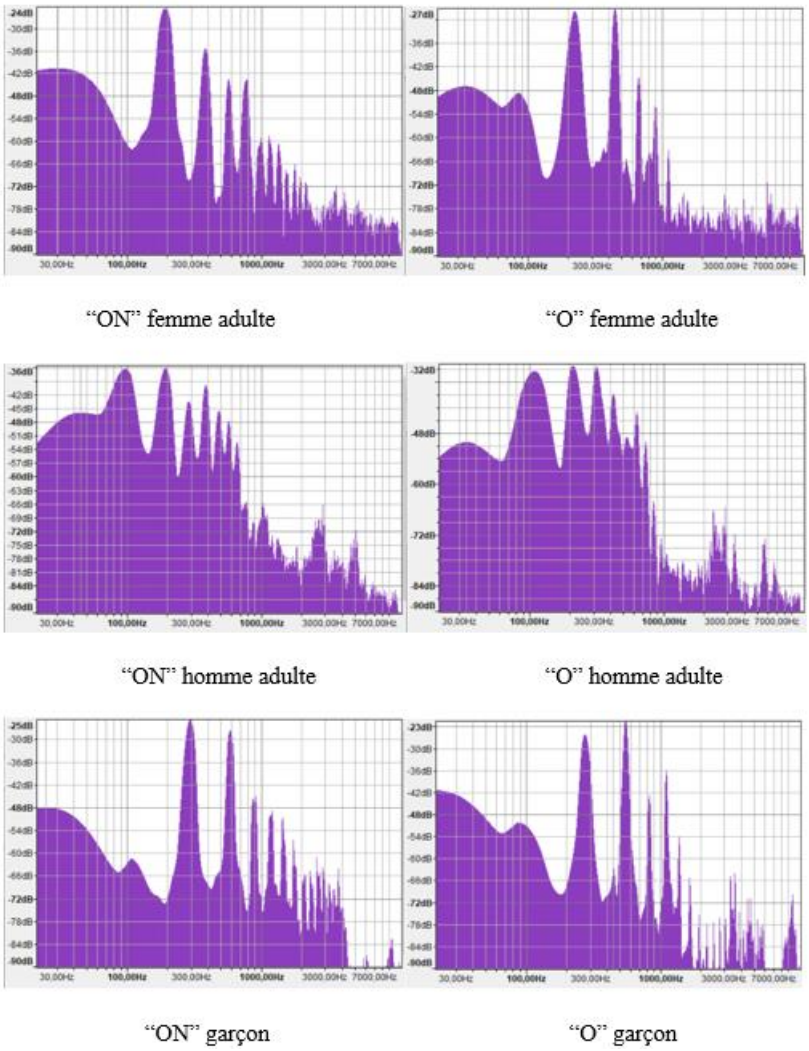

Figure 2 - Spectres des voyelles issues des enregistrements

A l'aide de ces données, on constate que chez les personnes adultes, la nasalisation s'accompagne d'un déplacement de la fréquence fondamentale dans les aigus de 20 Hz en moyenne alors que chez l'enfant c'est le contraire qui se produit. Cependant, un schéma général semble apparaître des figures ci-dessus: alors que pour le [O] la fréquence fondamentale et la 1ère harmonique sont d'intensité pratiquement égale dans les 3 cas avant une baisse linéaire pour les harmoniques suivantes, cette baisse "en escalier" s'exprime dès la 1ère harmonique pour le [Õ]. Ce schéma s'explique par la perte d'intensité de la 1ère harmonique lors de la nasalisation. En dehors de cela, les voyelles nasales et orales suivent les mêmes inflexions.

2. Un autre mécanisme rentre en jeu lors de la nasalisation

2.1 Expériences montrant le rôle du passage de l'air par le nez

Afin de déterminer le rôle du passage de l'air par le nez dans la nasalisation, nous avons décidé demander à 3 personnes de prononcer "O" puis "ON" en obstruant leurs conduits nasaux. Les spectres obtenus à l'aide d'Audacity (Audacity) sont présentés dans la figure 3.

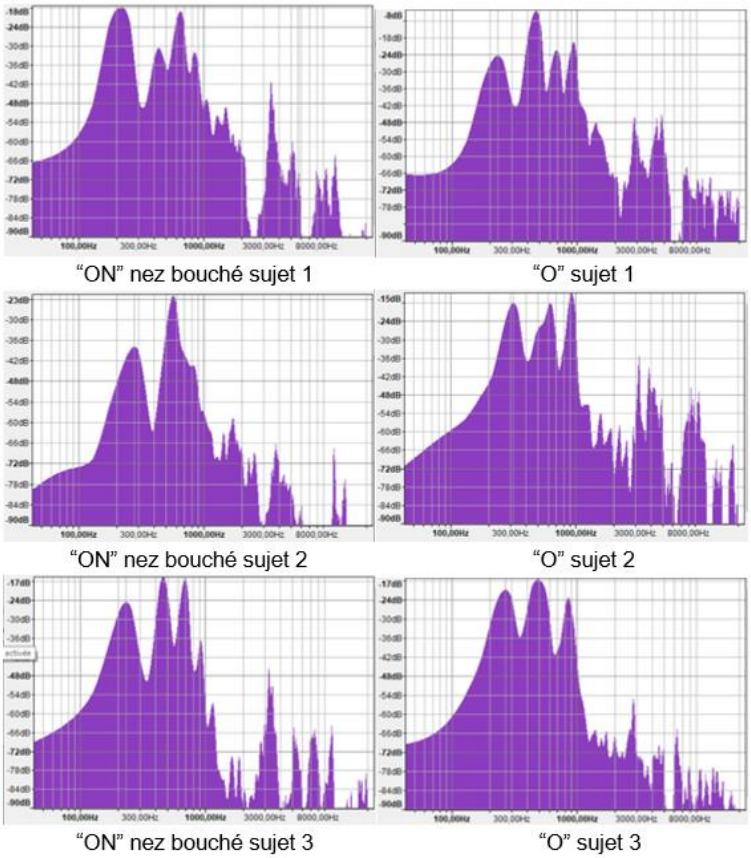

Figure 3 - Spectres des trois sujets

Pour les trois sujets, nous observons des différences dans le "O" et le "ON" avec les conduits nasaux obstrués. En effet, pour le "ON" avec le nez bouché, nous constatons un amortissement de l'intensité de la 1ère harmonique ainsi qu'un creux (une absence de son) autour de 3000 Hz. Ces différences nous amènent à nous demander si l'activité du *levator palatini*, c'est-à-dire le mécanisme d'abaissement du voile du palais permettent le passage d'air par le nez, est le seul mécanisme qui entre en jeu dans la nasalisation des sons.

2.2 L’ajustement de la cavité pharyngale

Dans la langue française, en plus du mécanisme d’abaissement du voile du palais qui crée l’amortissement de la 1ère harmonique comme vu précédemment, la nasalisation apparaît également par l’ajustement de la cavité pharyngale. Cela permet une annulation de certaines harmoniques expliquant le creux observé autour de 3000 Hz. C’est l’ajustement de cette cavité pharyngale qui nous permet de distinguer les différentes nasalités de la langue française. Les schémas de la figure 4 illustre la position de cette cavité pour différentes nasalités.

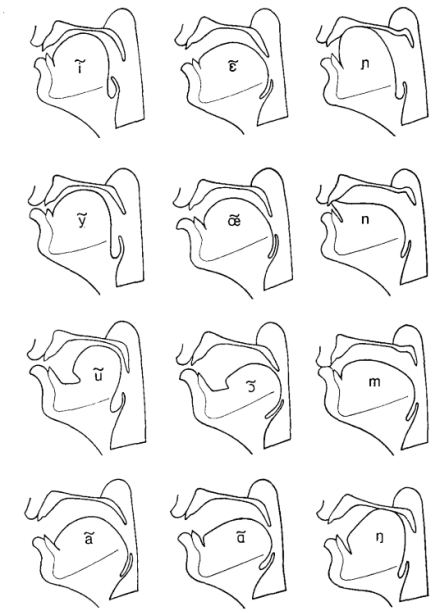

Figure 4 - Schémas illustrant différentes nasalités de la langue française

3. Reconnaissance d'une voyelle nasale à l'aide d'un réseau de neurones

Nous avons découvert les réseaux neurones et leur utilité dans le domaine de l'intelligence artificielle grâce à un projet réalisés dans l'UF *Initiation à Python* en 2ème année MIC à l'INSA de Toulouse. Lors de ce projet, il nous était demandé de réalisés en plusieurs étapes guidées un réseau de neurones dans le but « d'apprendre » à une machine à reconnaitre les chiffres de 0 à 9 dessinées dans une matrice à base de 0 et de 1, chaque chiffre pouvant être dessiné de façon différente. La figure 5 montre comment le même chiffre peut être dessiner de deux façons différentes.

```
nb0a = [[1,1,1,1,1],
        [1,0,0,0,1],
        [1,0,0,0,1],
        [1,0,0,0,1],
        [1,0,0,0,1],
        [1,1,1,1,1]]
```

```
nb0b = [[0,1,1,1,0],
        [1,0,0,0,1],
        [1,0,0,0,1],
        [1,0,0,0,1],
        [1,0,0,0,1],
        [0,1,1,1,0]]
```

Figure 5 - Deux façons différentes de dessiner un 0

A la fin du projet, la machine était effectivement capable de reconnaitre les chiffres de 0 à 9. Nous avons donc eu l'idée de reprendre cette méthode d'apprentissage afin de l'appliquer à la reconnaissance d'une voyelle nasale par rapport à sa sœur orale.

3.1 Généralités sur les réseaux de neurones

Le réseau de neurones artificiels est un système informatique dont le fonctionnement est similaire à celui des neurones du cerveau humain. C'est une variété de technologie de *Deep Learning* (apprentissage profond), une sous-catégorie du *Machine Learning* (apprentissage automatique) (Bastien, 2019). Dans la suite, afin d'en savoir en peu plus sur les réseaux de neurones artificiels, nous exposerons des parties du cours de Marc Parizeau sur les réseaux de neurones (Parizeau, 2004).

Histoire

Le cerveau d'un homme compte en moyenne 100 milliards de neurones qui lui permettent de réaliser tous les mécanismes essentiels au bon fonctionnement de l'organisme. Ces neurones sont organisés en réseau et reliés par des synapses. Cette organisation de la prise de décision a inspiré la plupart des architectures de réseau de neurones artificiels. Les premières recherches sur ces réseaux remontent à la fin du 19e siècle par des scientifiques comme Hermann von Helmholtz, Ernst Mach et Ivan Pavlov. Ce ne sont là que des théories générales sans l'interventions de modèle mathématique précis. Il faut attendre les années 1940 pour que des scientifiques comme Warren McCulloch, Walter Pitts et Donald Hebb montrent que de tels réseaux peuvent calculer n'importe quelle fonction logique et proposent une théorie pour l'apprentissage. La première utilisation concrète d'un réseau de neurones remontent à la fin des années 1950 avec le « *perceptron* » développé par Frank Rosenblatt. Ce réseau est capable de reconnaître des formes. Il a été démontré par la suite que des réseaux de la complexité du « *perceptron* » ne pouvait résoudre qu'une quantité limitée de problème. Cette découverte a ralenti les recherches sur le sujet jusqu'aux années 1980 où l'algorithme de rétropropagation des erreurs a été découvert qui

répond au critique faites aux réseaux de neurones. Depuis, de nouvelles théories, structures et algorithmes sont constamment développés.

Domaines d'application

Les réseaux de neurones sont utilisés de nos jours dans de nombreuses applications comme par exemples des systèmes d'autopilotage d'avion, de guidage pour automobile, de synthèse de la parole, de prévisions sur les marchés monétaires, le diagnostic médical et dans de nombreux autres domaines.

Le perceptron multicouche

Sans s'aventurer dans le formalisme mathématique du réseau perceptron, définissions ce qu'est ce type de réseau. Le perceptron multicouche est un réseau à propagation vers l'avant c'est-à-dire que l'information se propage de l'entrée vers la sortie, sans rétroaction. Il bénéficie d'un apprentissage supervisé par correction des erreurs, c'est-à-dire que les données d'entrées sont couplées avec leurs sorties désirées (appelées cibles). Cette méthode d'apprentissage est également appelée l'apprentissage par exemple. La correction des erreurs se fait à chaque itération de l'apprentissage.

Ce réseau de neurones et l'un des réseaux les plus utilisés pour des problèmes d'approximation, de classification et de prédiction. Il se compose d'entrées, de plusieurs couches cachées de neurones matérialisés par des matrices poids et des sorties. Dans la matrice poids, les poids dans la matrice de chaque neurone sont des scalaires qui sont modifiés lors de la phase d'apprentissage du réseau et qui, après apprentissage, permet de calculer la sortie en fonction de l'entrée.

3.2 Les objectifs de notre algorithme utilisant un réseau de neurones

Nous utiliserons donc le modèle du réseau de neurones perceptron pour faire la différence entre une voyelle orale et sa variante nasale. Les voyelles que l'on souhaitera différencier sont le ɔ̃ et o (« on » et « o »). L'entrée de l'algorithme sera le spectre du son de la voyelle prononcée. Ce spectre est récupéré sous la forme d'un tableau à deux colonnes, représentant les différente niveaux (en dB) en fonction de la fréquence (en Hz). Ce tableau est récupéré à l'aide du logiciel Audacity (Audacity). La sortie sera donc la voyelle identifiée par le réseau. Le premier objectif sera que le réseau puisse identifier correctement un spectre qu'il a déjà dans sa base de données, c'est-à-dire un spectre qu'il a déjà appris à reconnaître. Ensuite, le réseau devra reconnaître un spectre qu'il ne connaît pas. On se limitera à la reconnaissance de voyelles issues du même locuteur, les spectres d'un locuteur à l'autre étant trop différents pour être gérés par le modèle utilisé. Le réseau sera codé et compilé avec IDLE en Python 3.7.

3.3 Les étapes de réalisation du réseau de neurones

Le code complet et commenté de l'algorithme étant donné en annexe de ce rapport, il s'agira ici de décrire les grandes lignes de la réalisation du réseau de neurones ainsi que l'objectif des sous-programmes le composant, sans s'attarder à expliquer ligne par ligne ce que fait le code.

Structuration des données d'apprentissage

Les spectres des voyelles en entrée sont récupérés à l'aide de Audacity sous la forme de fichiers textes (.txt). Il fallait donc convertir les fichiers textes en objet exploitable et modifiable facilement en python. Nous avons donc choisi de ranger les niveaux et les fréquences dans deux sous-listes d'une même liste. Ces données peuvent donc être parcourues itérativement à la façon d'un tableau à deux entrées ou d'une matrice. La lecture du fichier texte et le stockage dans le tableau se fait à l'aide des méthodes *.readlines()* et *.append()*. Après avoir stocké les données dans des tableau, il y a une série de transformation à effectuer le tableau afin que l'apprentissage se passe dans de bonnes conditions.

Tout d'abord, les spectres enregistrés sont composés de niveaux négatifs en dB. Or, l'apprentissage du réseau se fera à l'aide d'un produit scalaire entre l'entrée et la matrice poids, tout cela avec un facteur 0, -1 ou 1 en fonction du poids que l'on souhaite donnée à un neurone. Donc si l'entrée comporte que des valeurs négatives, cela donnera lieu à des faux poids positifs. En effet, le produit de deux nombres positifs est négatif. De plus, les relations d'ordre des valeurs négatives sont inversées par rapport aux distances à zéro. Il faut donc travailler avec des valeurs positive afin les grandes valeurs en dB (les pics) puissent avoir un plus grand impact que les petites valeurs. Cette transformation est faite avec la fonction *ajuster()*.

Ensuite, lorsque qu'on observe la courbe des spectres avec la méthode *pyplot* de *matplotlib* on remarque une décroissance générale des niveaux. Cette décroissance générale peut être problématique car des pics qui se situeraient dans des hautes fréquences auront un poids moins important que des pics en basses fréquences. Il faut donc « redresser » la courbe. Pour cela, nous avons développé une fonction *redresser()* qui calcul la décroissance moyenne entre deux valeurs successives sur toute la courbe et qui ensuite à chaque niveau de rang *i* lui ajoute le produit cette décroissance moyenne avec le rang. La figure 6 illustre les changements effectués par la fonction

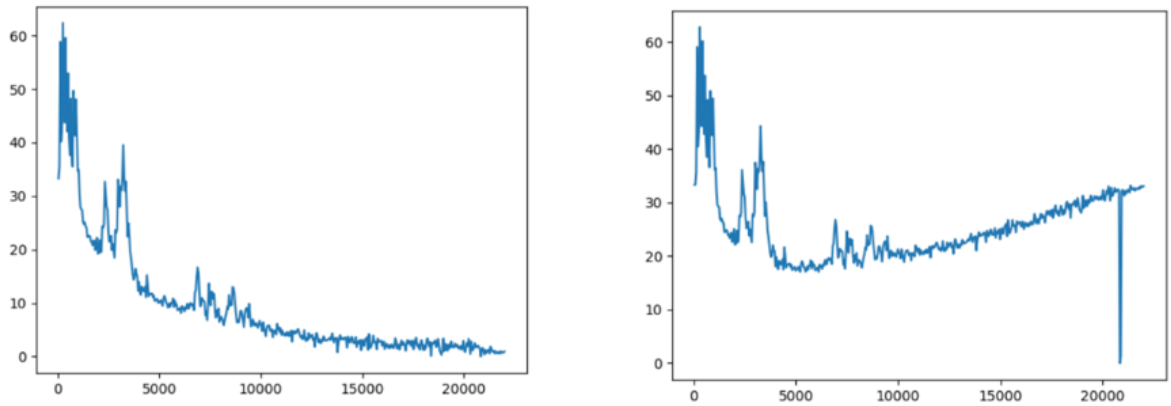

Figure 6 - Spectre avant (gauche) et après (droite) redressage

Nous avons ensuite remarqué que dans les spectres du locuteur des voyelles, les niveaux après 10 kHz ne sont plus significatifs. Afin qu'ils ne soient pas pris en compte lors de l'apprentissage, nous avons filtrer les fréquences, ne laissant que celles en dessous de 10 kHz et forçant les autres à 0. Nous avons donc codé une fonction *filtre_passe_bas()*.

Les spectres ont ensuite été réajuster et passer au filtre de Savitzky-Golay afin de lisser les courbes et laisser les pics apparents. Ce filtre est le seul filtre que nous n'avons pas codé nous-même, il a été récupérer sur *scipy.signal*. Il passe par l'approximation de la courbe par un développement limité à un ordre variable. Enfin, tous les spectres sont « normalisés » par rapport à leurs maximum respectif afin d'avoir des niveaux sur la même échelle. Cela néglige les effets des « j'ai parlé plus fort en enregistrant cette voyelle que celle-là ». Dans la figure 7, on remarque bien que les niveaux sont sur une échelle de 0 à 100.

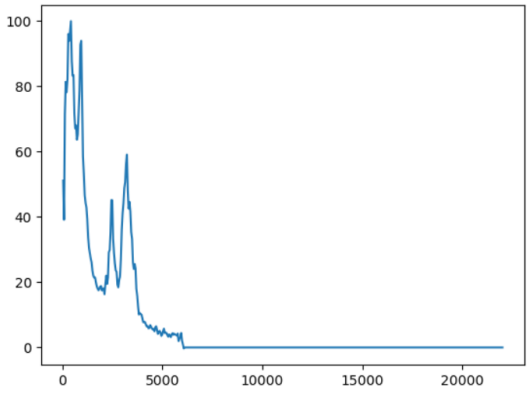

Figure 7 - Spectre après tous les traitements réalisés

Les données pour l'apprentissage sont ensuite rangées dans un dictionnaire.

Création de la matrice poids

La matrice poids va modéliser le poids accordé à chaque neurone lors de l'apprentissage afin de calculer la sortie. Les tableaux des spectres générés par Audacity contiennent 511 valeurs de niveaux, chacune associé à une fréquence. La matrice poids aura donc 511 neurones d'entrées et deux sorties afin d'identifier un « o » ou un « on ». Avant l'apprentissage, le poids de chaque neurone est généré aléatoirement entre -25 et 25.

Fonction d'activation des neurones pour la sortie

La fonction d'activation des neurones est la fonction qui permet de calculer la sortie du réseau en fonction de l'entrée et de la matrice poids. Cette fonction est modélisée par la fonction *calcul_neurone()*. Cette fonction réalise le produit scalaire de l'entrée avec la matrice poids puis retourne 0 si ce produit scalaire est négatif ou 1 sinon. C'est donc une fonction seuil. Ce produit scalaire est réalisé 2 fois pour chaque sortie du réseau. Il y a donc une valeur booléenne associée à la sortie « o » et un à la sortie « on ». Par exemple, lorsqu'un « o » est identifié, la fonction d'activation devrait retourner une liste de la forme [1, 0]. Les cas [1, 1] et [0, 0] sont donc des cas d'indéterminations.

Algorithme d'apprentissage du réseau

L'apprentissage du réseau se fait avec la fonction *apprendre().* Cette fonction modifie le poids des neurones dans la matrice poids en fonction de la valeur de sortie désirée et la valeur réelle calculé par *calcul_neurone().* La formule exacte est :

*poids = poids + (valeur_desiree - valeur_calculee) * e * h*

poids est le poids du neurone, *e* est la donnée et *h* un coefficient d'apprentissage qui permet d'accélérer l'apprentissage mais s'il est trop grand il présente des risques de divergence d'un poids qui ne devrait pas être aussi grand.

On remarque que le terme *(valeur_desiree - valeur_calculee)* peut prendre trois valeurs :

- **0** si la valeur calculée et la valeur désirée sont identiques, le poids n'est donc pas modifié

- **-1** si la valeur désirée est 0 mais la valeur calculée est 1, le poids est donc diminué afin d'arriver à un poids négatif. C'est pour cela que les données *e* doivent être positives

- **1** si la valeur désirée est 1 mais la valeur calculée est 0, le poids est donc augmenté afin d'arriver à un poids positif.

L'apprentissage doit s'effectuer un certain nombre de fois afin que la matrice poids soit fonctionnelle, on définit ce nombre d'occurrences avec la variable *nb_rounds.*

Algorithme de test du réseau

Afin de tester ce réseau, nous avons réalisé une fonction de test qui compte le nombre de réussites, d'échecs et d'indéterminations à la reconnaissance d'un son sur un certain nombre de calculs de la sortie. Une nouvelle matrice poids est générée à chaque itération de cette fonction de test car, pour une même matrice poids et pour une même entrée, la sortie calculée sera toujours la même. Cette fonction de test s'appelle *performance.* Elle prend en argument le spectre d'entrée, la sortie attendue, la matrice poids à modifier et le nombre de tests à effectuer.

3.4 Les résultats de la reconnaissance à l'aide de cet algorithme

Dans le dictionnaire d'apprentissage, on place un spectre par sortie, c'est-à-dire un spectre « o » et un spectre « on ». Ces spectres sont appelés respectivement *o_data* et *on_data*. Les spectres extérieurs au dictionnaire d'apprentissage sont également générés et son appelés *o* , *o_test* , *on* et *on_test* . Tous les spectres utilisés viennent d'enregistrements du même locuteur.

Reconnaissance d'enregistrements présents dans le dictionnaire d'apprentissage

Nous souhaitons tester si le réseau est capable de reconnaître les spectres des voyelles qu'il a déjà appris. Nous exécutons donc le test de performance sur les spectres *o_data* et *on_data*. Nous obtenons les résultats suivants exposés dans la figure 8.

```
>>> performance(o_data, [1, 0], poids, 30)
reussites: 30 , echecs: 0, indetermines: 0
>>> performance(on_data, [0, 1], poids, 30)
reussites: 30 , echecs: 0, indetermines: 0
```

Figure 8 - Résultats des tests sur les spectres présents dans le dictionnaire d'apprentissage

Nous remarquons donc que le réseau arrive parfaitement à faire la différence entre les spectres présent dans son dictionnaire d'apprentissage.

Reconnaissance d'enregistrements extérieurs au dictionnaire d'apprentissage

Nous allons désormais tester si le réseau est capable de reconnaître des spectres de voyelles qui ne sont pas présent dans le dictionnaire d'apprentissage. Nous exécutons le test de performance sur les spectres *o* , *o_test* , *on* et *on_test.* Nous obtenons les résultats suivants exposés dans la figure 9.

```
>>>performance(o, [1, 0], poids, 30)
reussites: 29 , echecs: 0, indetermines: 1
>>>performance(o_test, [1, 0], poids, 30)
reussites: 17 , echecs: 2, indetermines: 11
>>>performance(on, [0, 1], poids, 30)
reussites: 28 , echecs: 0, indetermines: 2
>>>performance(on_test, [0, 1], poids, 30)
reussites: 29 , echecs: 0, indetermines: 1
```

Figure 9 - Résultats des tests sur les spectres extérieurs au dictionnaire d'apprentissage

Les résultats sont plutôt concluants. On note quand même les 11 cas d'indéterminations sur *o_test*, ce qui nous amène à discuter des limites de l'utilisation de ce réseau de neurones.

3.5 Les limites de l'utilisation du réseau de neurones développé

Lors des tests, nous avons remarqué de nombreux cas d'indéterminations sur certains spectres, comme pour le spectre *o_test*. Cela traduit les limites d'utilisations de cette méthode pour la reconnaissance de voyelles nasales.

Les spectres issus d'un même locuteur peuvent présenter quelques variations de la position des pics significatifs de la voyelle ce qui peut induire la reconnaissance en échec ou en indétermination.

Parallèlement, les spectres varient en fonction de l'âge, du sexe et de l'accent du locuteur ainsi qu'en fonction d'autres paramètres physiologiques liés à la forme de la cavité pharyngale et du *levator palatini*. Cela modifie fortement l'allure des spectres et donc la position des pics significatifs, ce qui peut engendrer d'un locuteur à un autre des indéterminations et des échecs. En cas de changement de locuteur, il faudra très probablement changer certains des paramètres des filtres que subissent les données avant exploitations, comme par exemple modifier la fréquence de coupure du filtre passe-bas ou encore l'ordre du polynôme du filtre de Savitzky-Golay.

Plus généralement, les réseaux de neurones de type perceptron qui possèdent qu'une seule couche cachée de neurones comme celui que nous avons développé sont utile pour résoudre des problèmes assez simples. La reconnaissance de matrices représentant les chiffres de 0 à 9 se fait parfaitement bien avec ce type de réseaux. Lorsque qu'on regarde l'allure générale des spectres que nous proposons au réseau, on remarque qu'elles sont à l'œil nu très ressemblantes. Le réseau arrive tout de même sur plusieurs occurrences à reconnaitre plus souvent la bonne voyelle. Cependant, dès que la voyelle est prononcée un peu différemment par le locuteur, ou lorsque qu'on change de locuteur, la reconnaissance devient alors impossible. Pour pallier ces problèmes, il faudrait par exemple complexifier le réseau en augmentant le nombre de couches de neurones et inclure des algorithmes puissants de correction des erreurs comme par exemple l'algorithme de rétropropagation du gradient stochastique (Parizeau, 2004).

Conclusion

Ce projet a été pour nous l'occasion de commencer à nous approprier un phénomène qui nous était inconnu et pourtant auquel on fait appel quotidiennement. Cela nous a permis d'en apprendre un peu plus sur notre corps. De plus, les circonstances dans lesquelles une partie du projet a été réalisée nous ont également appris à travailler ensemble, d'une nouvelle manière, sans pouvoir être en contact direct. Ce contexte a pu en effet engendrer des difficultés niveau de la communication.

Nous avons abordé ce phénomène en plusieurs étapes. Commençant par la perception de ce qu'est la nasalisation, à l'oreille et dans au niveau de notre voile du palais. Puis, grâce au logiciel Audacity, nous avons abordé les caractéristiques d'une voyelle nasale qui permet de percevoir ce son. Cela nous a aider à comprendre et à illustrer les mécanismes physiologiques qui permettent la nasalisation. Enfin, après avoir abordé le mécanisme sur ces 3 aspects, le développement de l'algorithme nous a permis de mettre en pratique les notions algorithmiques acquises depuis la 1ère année, d'utiliser un nouveau langage de programmation et de mettre un premier pas dans le monde du *Machine Learning* avec une application directe de l'utilisation d'un réseau de neurones.

Bibliographie

Audacity. (s.d.).

Bastien, L. (2019, avril 5). *Réseau de neurones artificiels : qu'est-ce que c'est et à quoi ça sert ?* Récupéré sur Le Big Data: https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition

Nasalisation. (s.d.). Récupéré sur Wikimonde.

Parizeau, M. (2004). Réseaux de neurones GIF-21140 et GIF-64326. Université Laval.

Vaissière, J. (1995). Nasalité et phonétique.

Delvaux, V., Metens, T., & Soquet, A. (2002). Propriétés acoustiques et articulatoires des voyelles nasales du français. XXIVèmes Journées d'étude sur la parole, Nancy, 1, 348-352.

Table des illustrations

Tables des Annexes

1- TABLEAUX RÉCAPITULATIFS DE L'ANALYSE DES SPECTRES ISSUES DU CORPUS FOURNI PAR NOTRE TUTEUR

"ON" femme adulte	Fondamentale (193Hz)	1ere harmonique	2ème harmonique	3ème harmonique	4ème harmonique
Intensité sonore (dB)	**-24**	**-34**	**-39**	**-43**	**-55**

"O" femme adulte	Fondamentale (220Hz)	1ere harmonique	2ème harmonique	3ème harmonique	4ème harmonique
Intensité sonore (dB)	**-26**	**-27**	**-44**	**-50**	**-64**

"ON" homme adulte	Fondamentale (95Hz)	1ere harmonique	2ème harmonique	3ème harmonique	4ème harmonique
Intensité sonore (dB)	**-34**	**-33**	**-41**	**-36**	**-39**

"O" homme adulte	Fondamentale (110Hz))	1ere harmonique	2ème harmonique	3ème harmonique	4ème harmonique
Intensité sonore (dB)	**-33**	**-32**	**-32**	**-39**	**-49**

"ON" garçon	Fondamentale (296Hz)	1ere harmonique	2ème harmonique	3ème harmonique	4ème harmonique
Intensité sonore (dB)	**-24**	**-27**	**-45**	**-49**	**-50**

"O" garçon	Fondamentale (274Hz)	1ere harmonique	2ème harmonique	3ème harmonique	4ème harmonique
Intensité sonore (dB)	**-25**	**-21**	**-43**	**-36**	**-53**

2- CODE PYTHON DE L'ALGORITHME

```
import numpy as np

from math import exp

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import savgol_filter


## les sons doivent être recuperées à l'aide du fichier txt produit par la

## le spectre de Audacity


## Fonction pour transformer le fichier .txt en numpy array Frequence, Niveau

## Le fichier .txt doit être dans le repertoir courant, les virgule doivent etre en point pour le type float


def txt_to_array(file):

    spectre = open (file)

      ligne1 = spectre.readline() ## Ne pas traiter la premiere ligne qui ne contient pas de val

      lignes = spectre.readlines() ## recuperation des lignes [ chaque ligne est comme suit " Freq Niveau " ]

      freq = []

      niveau = []


      for l in lignes:

        lsplit = l.split() # Separation de la ligne en deux case de liste [ 0 => Freq ; 1 => Niveau ]

      freq.append(float(lsplit[0])) # list de freq

        niveau.append(float(lsplit[1])) # liste de niveau

      a = [freq,niveau] #crée un array Freq , niveau

      spectre.close()

      return a
```

```
## fonction pour decalé les niveau en dB initalement negative ###

## on recherche le niveau min du tableau, et on ajoute la valeur absolue de ce niveau ##

def ajuster(array):

      minimum = max(array[1]) + 1.0

      for i in range(len(array[1])):

      if array[1][i] != 0.0 and array[1][i] < minimum :

            minimum = array[1][i]

      for i in range(len(array[1])):

      if array[1][i] !=0:

            array[1][i] -= minimum

      return array

### fonction pour exponentié l'array de sorte à faire apparaitre les niveau significatif

def exponentier(array):

      for i in range(len(array[1])):

      array[1][i] = exp(array[1][i])

      return array

## algorithme pour que les données soit reduit à la même echelle

def normaliser_Moy(array):

      Moy = 0.0

      for i in range(len(array[1])):

      Moy += array[1][i] / len(array[1])

      for i in range(len(array[1])):

      array[1][i] = (array[1][i] / Moy) * 100

    return array

def normaliser_Max(array):

      maximum = max(array[1])

      for i in range(len(array[1])):

      if maximum != 0:
```

```
            array[1][i] = (array[1][i] / maximum)*100
      return array


def redresser(array): ## algorithme pour annuler l'effet de decroissance générale des niveau
      new_array = [array[1][0]]
    delta = 0
    card = len(array[1])
    for i in range(card-1):
      if array[1][i]*array[1][i+1] == 0.0:
            card -= 1
      else:
            delta += (array[1][i]-array[1][i+1]) / (card - 1)
    for k in range(1,len(array[1])):
      if array[1][k]*array[1][k-1] != 0.0:
            new_array.append((array[1][k-1]) + delta*k )
      else:
            new_array.append(array[1][k])
      return [array[0],new_array]


def filtre_passe_haut(array,wc): ## wc est la frequence de coupure du filtre
      k = 0
      nb = 0
      while array[0][k] < wc:
      nb += 1
      k += 1
      for i in range(len(array[1])-(len(array[1])-nb)):
      array[1][i] = 0.0
      return array
```

```
def filtre_passe_bas(array,wc): ## wc est la frequence de coupure du filtre
      for i in range(len(array[1])):
      if array[0][i] > wc:
            array[1][i] = 0.0
      return array


def filtre_passe_bande(array,wc1,wc2):
      return filtre_passe_bas(filtre_passe_haut(array,wc1),wc2)


def txt_to_array25(file):
    a =
ajuster(filtre_passe_bas(redresser(ajuster(txt_to_array(file))),6000.0))
      return normaliser_Max(savgol_filter(a,5,3))


####### début du réseau de neuronne #############


##### Données pour l'apprentissage , le spectre est une array de 2,511


## tableau o :
o_data =  txt_to_array25("spectre_o_data_seul_raphael.txt")
o =  txt_to_array25("spectre_o_seul_raphael.txt")
o_test = txt_to_array25("spectre_o_test_seul_raphael.txt")
##tableau on  :
on_data = txt_to_array25("spectre_on_data_seul_raphael.txt")
on = txt_to_array25("spectre_on_seul_raphael.txt")
on_test = txt_to_array25("spectre_on_test_seul_raphael.txt")
##dictionnaire des données : 0 sont les o et 1 les on
d = { 0 : [o_data] , 1 : [on_data] }
```

```
##### Création du réseau ############ type MLP


### Surcouche pour crée une matrice poids au hasard ##

from numpy import random

def random_within(a, b):

      return (b-a) * random.random() + a

def random_list(n, a, b):

      return [random_within(a, b) for i in range(n)]

def init_poids(dimEntree, dimSortie):

      poids = [random_list(dimSortie, -25, 25) for i in range(dimEntree)]

      return poids


### La matrice poids se définie : Dim entrée => poids de chaque Niveau
[511]

###                            Dim sortie => nombre de sons à reconnaitre
[2]


##### Fonction de combinaison et fonction d'activation ##############

import math

## On chosis un reseau de type MLP (multi-layer perceptron), la fonction de
combinaison

## est le produit scalaire entre les vecteurs d'entrées et les vecteurs de
poids synaptiques


def calcul_neurone(j, e, poids):

## j est le numero du son à reconnaitre , e est l'array du son

      resultat = 0.0

      count = 0

      for i in range (len(e[1])):

        resultat = resultat + e[1][i] * poids[i][j]

      if resultat > 0:

        boolean = True
```

```
        else:

          boolean = False

        seuil = int(boolean) ## Correspond a la fonction d'activation [seuil
0 1]

        return seuil


def calcul_reseau(e , poids ): ##Cette fonction va passer la fonction de
combinaison aux neronnes des deux sons

        resultats = [calcul_neurone (i, e, poids) for i in range(2)]

        return resultats


##### Apprentissage du reseau ######

## On va forcer la sortie d'un reseau en fonction de l'entrée et de la
sortie attendue


def apprendre_neurone(e, poids, j, sortie_attendue):

        h = 10 ## ceci est le parametre d'apprentissage, a modifier en cas de
divergence de l'apprentissage

        valeur_calculee = calcul_neurone ( j, e, poids)

        for i in range (len(poids[0])) :

          valeur_desiree = 0

        if sortie_attendue == j:

              valeur_desiree = 1

        poids[i][j] = poids[i][j] + (valeur_desiree - valeur_calculee) *
e[1][i] * h

        return poids


def apprendre_reseau(e, poids, sortie_attendue):

        for j in range(2):

        poids = apprendre_neurone(e, poids, j, sortie_attendue)

        return poids
```

```
def apprendre(d, poids):
      for k in d:
      for e in d[k]:
            poids = apprendre_reseau(e, poids, k)
      return poids


##### Test du réseau #########


## on doit mettre un son en entrée comme décrit en haut
son = o_test
def performance(son,attendu,poids,rounds):
      echec = 0
      reussite = 0
    indetermine = 0
      for i in range(rounds):
      poids = init_poids( 511 , 2 )


        nb_rounds = 100 ##nombre d'occurences d'apprentissage


      for i in range(nb_rounds):
            poids = apprendre(d, poids)
        resultat = calcul_reseau(son, poids) # on passe le son dans la
matrice poids
      if resultat == attendu:             # on compte le nombre de
reussite, d'echec et d'indetermination
            reussite += 1
      elif resultat == [1, 1] or resultat == [0,0]:
            indetermine +=1
      else:
            echec += 1
```

```
      a = "reussites: {} , echecs: {}, indetermines:
{}".format(reussite,echec,indetermine)

      print (a)

poids = 0

performance(son,[1, 0],poids,30)


## Affichage de courbes ###

x = np.array(o[0])

y = np.array(o_test[1])

y2 = np.array(on_data[1])


plt.plot(x , y)

plt.plot(x , y2)

#plt.show()
```

INSA Toulouse
135, avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Université Fédérale Toulouse Midi-Pyrénées

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE