Détection de marqueurs discursifs par analyse de spectrogramme via un réseau neuronal convolutif.

La tâche d'identification de marqueurs discursifs (dm) a pris différentes formes tout au long des projets de recherche ayant été menés à ce sujet (voir les articles compilés dans le google drive). Ces différentes approches peuvent être intégralement "lexicales" : elles n'utilisent que la retranscription d'un discours pour détecter les dm grâce au posTag des mots de son contexte, par exemple. Il s'agit là de l'approche la plus performante aujourd'hui. Il est également possible d'augmenter la précision de tels systèmes en employant des paramètres supplémentaires tels que certains indices sociolinguistiques (âge / origine / catégorie socio-économique du locuteur, contexte et but du discours…) ou plutôt prosodiques (longueur du mot, longueur des pauses autour du mot, F0, lieu d'accentuation, etc.). Cette dernière méthode ne semble pas vraiment permettre une détection efficace des dm à elle seule mais devient intéressante en complément de la première pour répondre à certains cas trop ambigus.

Parallèlement à la recherche menée sur l'identification des dm (que vous connaissez probablement mieux que moi), on observe aujourd'hui une certaine tendance dans la recherche et l'analyse de discours oraux à l'utilisation d'algorithmes de deeplearning et notamment aux CNN.
Ces "réseaux de neurones convolutifs" sont des algorithmes fonctionnant sur la base de réseaux de neurones classiques que certaines spécificités supplémentaires rendent très performants dans l'analyse d'images (voir la page wikipedia, très bien documentée et plutôt facile à comprendre, pour plus d'informations).
Il est donc tout à fait logique que leur succès dans ce dernier domaine ait inspiré certains linguistes à l'origine d'architectures de CNN permettant de faire de l'analyse de spectrogrammes. Aujourd'hui, l'utilité et l'efficacité des CNN dans ce genre de tâches n'est plus à prouver, comme le témoignent de nombreux projets de recherche ayant donné de très bons résultats sur tout un panel de tâches de classification liée au traitement automatique de la parole.

En résumé, nous avons vu que les méthodes de détection purement prosodique de dm, fonctionnant par analyse de certaines paramètres dont la F0, les pauses avant et après le mot, etc. ne donnent pas de très bons résultats (surtout si on les compare à l'analyse purement lexicale ou par posTag). Notre hypothèse pourrait donc être la suivante : est-ce que l'inefficacité de telles architectures (détection prosodique classique) serait simplement due à une mauvaise sélection des paramètres prosodiques analysés, auquel cas une architecture par CNN (dont les paramètres sont sélectionnés automatiquement sur tous les indices prosodiques disponibles sur un spectrogramme) montrerait des performances plus importantes. En d'autres termes, faire tourner un CNN sur une collection de spectrogrammes représentant la prononciation d'un mot donné (qu'il soit sous une forme dm ou non-dm) pourrait potentiellement surpasser les performances d'un classifieur plus classique qui ne se concentrerait que sur certains paramètres prédéfinis (F0, durée, etc.).
Dans le cas d'un succès du CNN (ou au moins d'un dépassement des performances de la baseline que serait ici la classification prosodique classique mise en oeuvre par plusieurs projets de recherche), les conclusions quant au marquage prosodique et phonétique des dm serait indéniable. De plus, une analyse plus poussée de la "manière" dont un tel CNN sélectionne certaines parties du spectrogramme dans son processus de détection pourrait nous donner de nombreuses informations supplémentaires.

J'ai donc tenté de mettre en place, à partir de cette intuition de avec Yaru Wu, une ébauche de CNN, dans l'espoir d'initier une certaine réflexion sur ces différentes hypothèses.

Voilà donc un compte rendu des différentes composantes techniques de cette ébauche d'architecture et une analyse succincte des résultats obtenus.

1. L'extraction de données sur Praat.

Un CNN, comme tout classifieur, a besoin d'un maximum de données d'entraînement pour proposer des résultats intéressants (si tant est que ces données soit analysables en soi). Il a donc fallu trouver un moyen d'extraire, à partir du corpus LOCAS que l'on m'a fourni, un ensemble de prononciations de mots. Pour cela, un script Praat (disponible dans le drive) m'a permis d'extraire toutes les occurrences d'un mot dans l'intégralité du corpus. J'ai conçu ce dernier de la manière suivante :

- accès à un dossier contenant tous les fichiers audio
- ouverture de chaque fichier audio et du textgrid correspondant
- parcours de l'audio et sélection de chaque occurence du mot recherché (tier 4 du textgrid)
- extraction de l'audio pour chaque occurence
- exportation de cet audio en .wav
- le nom donné au fichier .wav dépend du tier 8 du textgrid (dm ou autre chose)
- stockage de chaque fichier audio extrait dans un dossier

2. Fonctionnement du CNN

Pour des raisons pratiques, j'ai écrit le programme sur google-colab. Cela présente certains avantages, tels que la possibilité de partager le code facilement et d'utiliser des GPU de google en cas d'entraînement sur un trop grand corpus d'exemples (ce qui peut rapidement devenir compliqué au niveau algorithmique et demander des performances importantes à l'ordinateur).

Le code est inspiré en grande partie d'un CNN écrit par Rishi Sidhu et destiné à une tâche relativement différente : la reconnaissance de chiffres dictés. Une grande partie du code source était donc inutile (par exemple, les outils permettant d'enregistrer la voix de l'utilisateur en temps réel et de donner le résultat de la prédiction du chiffre prononcé). En revanche, les fonctions permettant de créer des spectrogrammes ont été gardées puisqu'elles proposaient un mécanisme très similaire à ce que demandait notre tâche.

3. Création des spectrogrammes

La première subtilité s'articule lors de la création des spectrogrammes : python dispose de packages permettant de générer des spectrogrammes très simplement à partir de fichiers .wav. Il est possible de générer une image à proprement parler (.png en couleur ou non). Il est également possible, comme je l'ai fait, de se contenter d'une matrice numpy qui vient stocker les différentes informations de couleur de cette image. Ce procédé présente alors plusieurs avantages : une matrice est beaucoup plus légère et demande moins de mémoire à l'ordinateur lors de son analyse qu'une image normale. De plus, l'algorithme du CNN demande la transformation des images en valeurs numériques analysables. Ici, cette étape est ignorée puisque les spectrogrammes sont directement générés sous forme de matrices de valeurs, chaque composante représentant un pixel par sa valeur en niveau de gris (les spectrogrammes étant initialement en couleur RGB, les valeurs sont transformées pour représenter ces niveaux de gris).

4. Entraînement de l'algorithme

On emploie alors le package Keras qui permet de créer des réseaux de neurones en précisant directement l'architecture désirée. Ici, on aura :

- Une première couche de convolution (activation ReLu, 32 filtres)
- Une seconde couche de convolution (activation ReLu, 64 filtres)
- Une couche de pooling

- Une première couche entièrement connectée (ReLu)
- Une seconde couche entièrement connectée (Softmax)

Cette architecture est relativement basique et semble recommandée pour le traitement de spectrogrammes. En revanche, j'admet ne pas avoir une connaissance assez étendue du fonctionnement des CNN et de Keras pour m'être aventuré sur d'autres architectures. En jouant un peu avec ces paramètres, j'ai néanmoins remarqué que ces derniers semblaient les plus efficaces (malgré les résultats plutôt mauvais de l'algorithme).

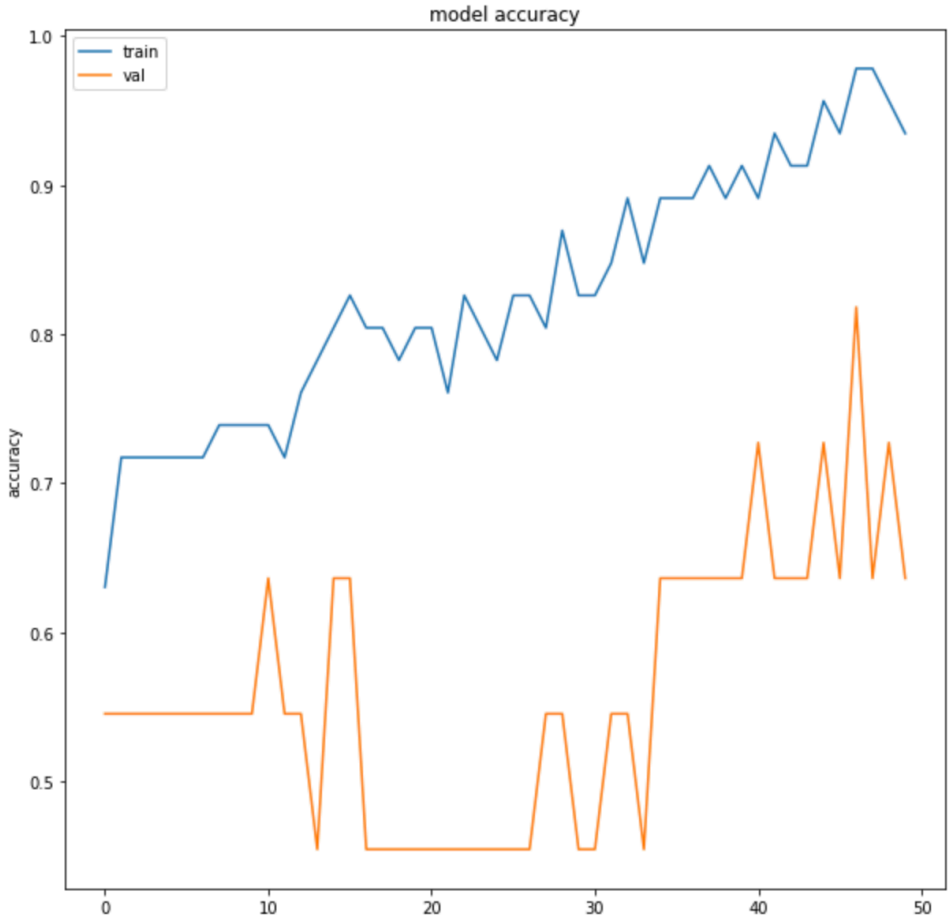
Le programmes va donc commencer par partager les exemples en un corpus de train et un corpus de test (environ 80% et 20%). L'entrainement se fait assez rapidement mais serait très probablement plus lent si le nombre d'exemples était plus élevé (ce qui donnerait aussi de meilleurs résultats).

5. Résultats

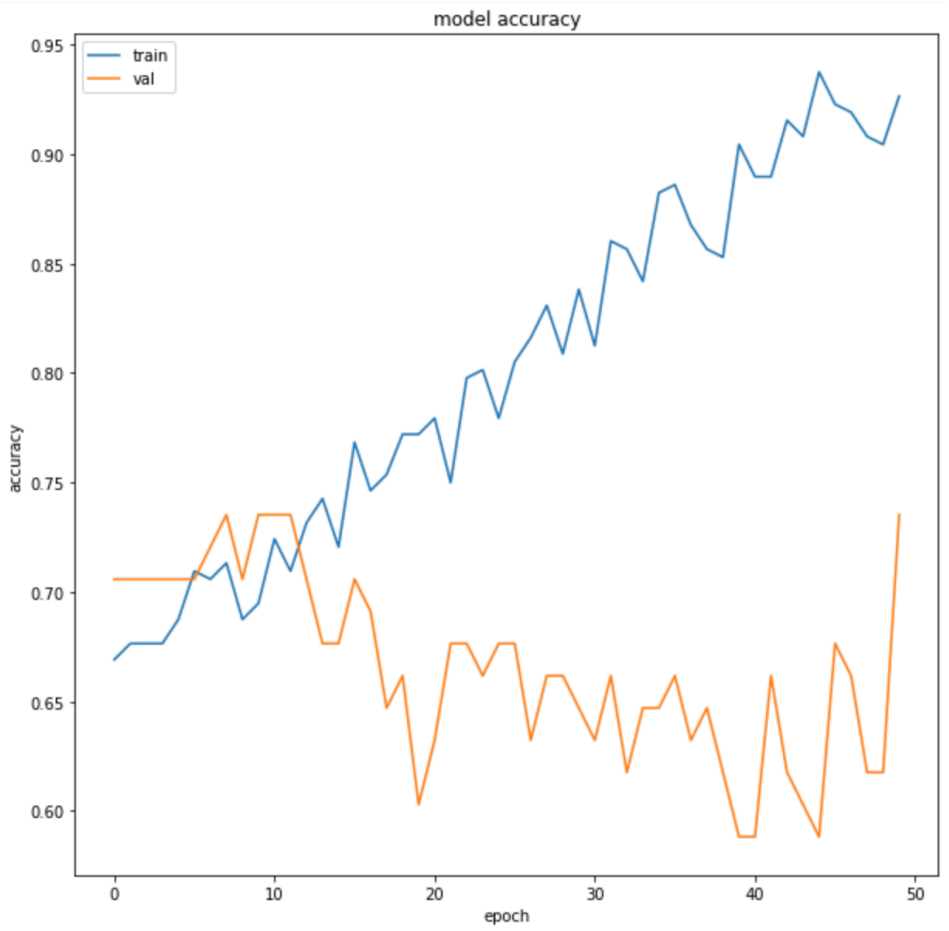
Les résultats obtenus ne sont pas très lisibles pour différentes raisons : les performances sont un peu chaotiques et évoluent parfois très aléatoirement au cours de l'apprentissage et je ne disposait pas d'assez de données pour pouvoir entrainer puis tester l'algorithme de manière optimale.

Le programme affiche tout simplement un ensemble de performances finales mais aussi leur évolution tout au long de l'apprentissage. Cette évolution est représentée par un graphique qui prend la forme suivante : des valeurs d'accuracy (proportion d'exemples correctement classés) pour le train en bleu et pour le test en orange.

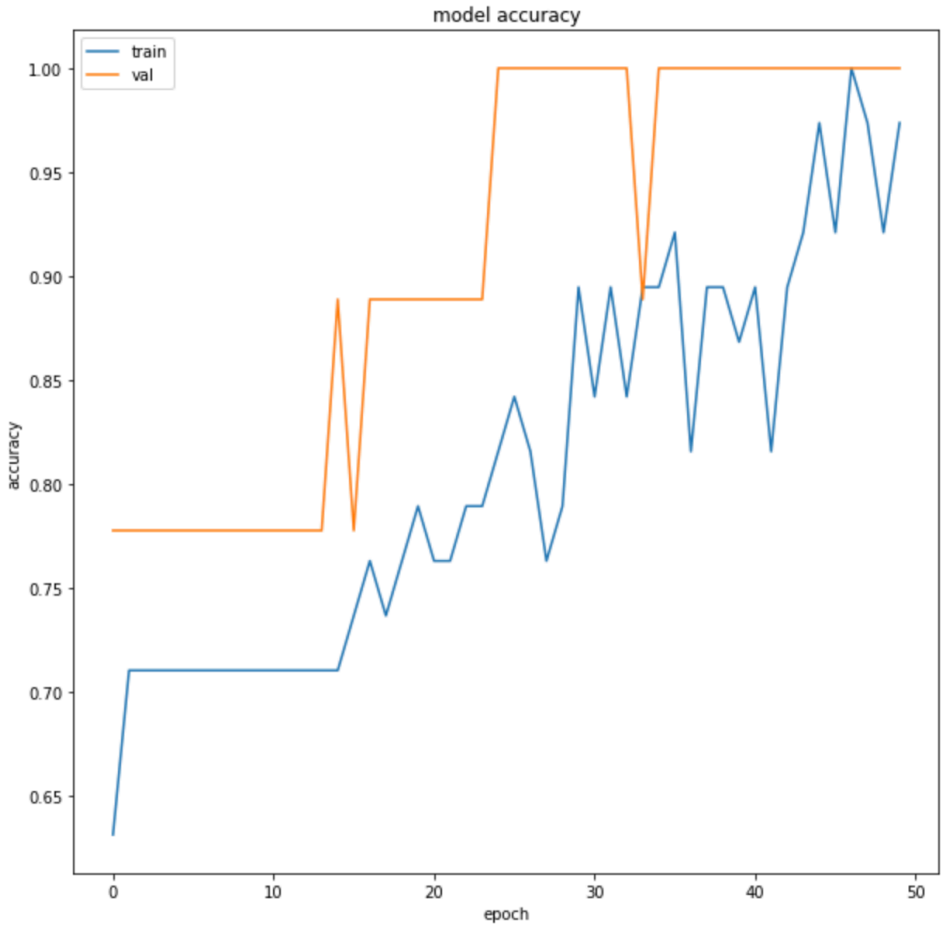
- pour le mot "bon" (57 exemples) :



- pour le mot “et” (341 exemples) :



- pour le mot “enfin” (47 exemples) :



J'ai choisi de baser mes explications sur ces mot d'une part parce qu'ils sont plus largement représentées dans le corpus ("et" étant le potentiel dm le plus fréquent) et parce qu'ils étaient les mieux découpés et les plus faciles à reconnaitre visuellement. Le mot "enfin", par exemple, est souvent prononcé différemment selon qu'il est un dm ("…fin") ou autre ("enfin"). J'ai donc pensé que l'algorithme aurait plus de facilité à classifier ces trois mots que les autres pour lesquels les performances sont encore plus chaotiques.
En effet, lors de certains entraînements sur "enfin", l'algorithme atteint 100% de précision sur le corpus de test (comme sur le graphique présenté ci-dessus). Il est tout de même important de rappeler que les 47 exemples sont partagés aléatoirement en 9 exemples de test et 38 de train… Ces données étant très petites, il est fort possible que les données de test soient intégralement d'une classe ou d'une autre (environ 9 ou 8 dm pour 0 ou 1 forme "non-dm"). L'algorithme est alors légèrement biaisé dans son calcul de performances puisqu'il risque de classifier les exemples de manière tout à fait naïve (bien que son accuracy sur les exemples de train augmente aussi de son côté). Il m'est donc très difficile d'interpréter des résultats qui ne sont pas vraiment constants et qui, comme pour les mots "et" et "donc", évoluent de manière un peu aléatoire sans montrer de réel apprentissage. On peut simplement conclure qu'une forme d'apprentissage et effectivement mise en place bien que rien ne nous indique si ce dernier est efficace ni sur quels paramètres il se base pour différencier les dm des non-dm. Il aurait donc été intéressant de générer des "heatmaps[1]" (au moins pour le mot "enfin") afin d'observer visuellement quelles sont les parties du spectrogramme que l'algorithme considère comme intéressantes dans sa classification.

Une dernière remarque avant de passer aux différentes hypothèses expliquant le dysfonctionnement apparent de l'algorithme : créer un CNN qui permettrait de classifier n'importe quel mot aurait été une tâche bien plus complexe et ne me semblait pas pertinent à ce stade. Les résultats sur des mots uniques étant encore trop flous pour conclure à une possibilité de reconnaissance des marqueurs discursifs par analyse de spectrogrammes, il est certain qu'un CNN entraîné sur une collection de mots différents n'aurait jamais fonctionné dans l'état. En revanche, cette piste de réflexion serait très intéressante pour, comme avec les algorithmes passant par des paramètres lexicaux, tenter de déceler une tendance plus générale dans les indices prosodiques propres marqueurs discursifs, quel que soit le mot employé.

Pour en revenir à nos résultats, différentes raisons peuvent expliquer leur étrangeté :

1. Les données ne sont pas suffisamment grandes : le corpus propose entre 50 et 100 occurrences pour la plupart des mots (sauf "et" qui en propose 341), ce qui, à mon avis, n'est tout simplement pas suffisant pour obtenir un CNN performant et efficace dans l'identification des dm.
2. Les mots que j'ai choisi sont ceux qui proposent le plus de formes "non-dm". Malheureusement, même avec ces mots-là, j'ai l'impression que la proportion de dm est souvent trop importante pour que l'algorithme ait accès à suffisamment d'informations quant à la différence entre les deux classes.
3. Les enregistrements ont des qualités trop différentes (certains sont plutôt propres et d'autres beaucoup plus bruités et de moindre qualité) ce qui peut potentiellement fausser les résultats puisque les spectrogrammes générés peuvent contenir beaucoup de "bruit" (surtout sur une si petite quantité de données).
4. L'annotation du corpus n'est pas toujours très précise et j'ai parfois dû enlever, à la main, les occurrences trop "décalées" : pour certaines occurrences, l'enregistrement ne correspond pas exactement au mot et n'a parfois rien à voir avec lui (ce qui est probablement dû à un mauvais découpage sur les textGrid). Ce dernier problème complique également l'ajout de données : si le CNN est entraîné sur des occurrences décalées et qu'il voit apparaitre des exemples qui ne correspondent pas du tout aux autres, il risque de ne pas fonctionner correctement. (Il existe en

---

[1] Une heatmap serait ici une image représentant, grâce à une couleur plus ou moins intense, par exemple, les différentes régions du spectrogrammes considérées comme "importantes" par l'algorithme dans sa classification : ainsi, les zones les plus marquées seraient celles auxquelles le CNN aurait donné le plus de poids et, par conséquent, le lieu où se situe l'indice prosodique marquant un dm.

revanche des méthodes de lissage permettant de supprimer les exemples trop différents pour ne garder que les plus cohérents).

6. Conclusion :

L'architecture de l'algorithme et le script d'extraction des occurrences de mots sur praat sont tous les deux opérationnels. On obtient des performances allant de 70 à 80% (voire 100% pour “enfin”) d’identifications correctes sur les petits corpus de test générés. Néanmoins, j’ai peur que cette performance (qui à l'air plutôt bonne en apparence) ne soit, en réalité, pas si intéressante, comme j’ai tenté de l’expliquer plus haut.

Je pense, en revanche, que l'idée reste bonne et qu'elle mériterait d'être approfondie avec de plus grandes données, des enregistrements / découpages plus précis, d’autres architectures de CNN et différents paramètres de génération des spectrogrammes. Il y a probablement une forme d'apprentissage qui se met en place mais je ne saurais dire sur quoi elle se base. Est-ce que l'algorithme détecte les variation du temps de prononciation, l'intensité de la première syllabe du mot, la pause qui le suit, la "troncation" du mot (pour "enfin" qui est souvent prononcé "fin" lorsqu'il est un dm) ou bien est-ce qu'il se base simplement sur une pondération un peu aléatoire qui "colle" bien au données fournies sans observer de réelles variations phonologiques entre les mots ?

Il me semble également primordial de souligner que l’échec d’une telle expérience, si elle était quelque peu approfondie et plus développée que l’ébauche que j’ai tenté de mettre en place, pourrait nous apprendre beaucoup sur la prosodie des marqueurs discursifs. Il est tout à fait possible qu’un tel algorithme ne puisse tout simplement pas fonctionner parce que la production d’un dm ne passerait pas, ou pas intégralement, par la prononciation du mot. C’est d’ailleurs, il me semble, ce que certains chercheurs ont pu remarquer en montrant que l’ajout d’informations prosodiques et phonétiques à des classifieurs lexicaux n’augmente que de très peu leurs performances. La production d’un dm ne passerait donc que très légèrement par sa prononciation qui ne se distinguerait pas assez de sa forme non-dm pour qu’un CNN puisse le reconnaître efficacement.
Il s’agit là d’une conclusion encore bien trop intuitive puisque mon algorithme peut tout à fait dysfonctionner pour d’autres raisons, peut-être plus banales et techniques. Je pense tout de même que la recherche en linguistique qui emploie des technologies propres au traitement automatique de la parole, et surtout des méthodes de classification aussi modernes que les CNN et l’analyse de spectrogrammes, se doit d’utiliser l’échec et les mauvaises performances de certains programmes comme une source d’apprentissage très fertile. Ici, prouver qu’un CNN ne peut pas distinguer correctement les spectrogrammes d’un dm et d’un non-dm permettrait de venir appuyer les conclusions d’autres projets de recherche qui se sont contentés de classifieurs plus classiques. Avec des programmes un peu plus poussés et notamment la génération d’heatmaps, on pourrait même montrer que les dm et les non-dm ne sont pas séparables en deux classes prosodiques distinctes mais constituent plutôt une évolution continue (un mot serait plus ou moins dm) dépendante du contexte d’apparition du mot. Ce dernier point de vue serait en réalité une très belle explication aux performances étranges que j’ai observées sur cet algorithme, bien que je ne puisse pas du tout me permettre de l’appuyer à ce stade.