De TOCQUEVILLE Adrien
FESSARD Grégoire

TIPE 2016 - 2017

RECONNAISSANCE VOCALE

RESEAUX DE NEURONES

SOMMAIRE

TABLE DES FIGURES

RESUME

Les systèmes de reconnaissance vocale permettent à l'homme de communiquer avec son ordinateur et ainsi faciliter et accélérer bon nombre d'actions. Chaque étape de ce processus sera décrite, depuis l'enregistrement et le traitement du signal afin de créer un vecteur caractéristique jusqu'à la reconstitution de phrase à partir d'une séquence de phonèmes sur laquelle on appliquera l'algorithme de Viterbi, d'un modèle de markov caché et l'utilisation de n-grammes. Pour cela nous ferons la description théorique de techniques de comparaison de motif, d'analyse de réseau de neurone et assisterons à la conception d'un réseau de kohonen à partir d'un vecteur caractéristique.

ABSTRACT

Speech recognition systems allow people to communicate with their computer and thus facilitate and accelerate several type of actions. Each step of this process will be described from the recording and processing of the signal to create a feature vector up to the reconstruction of a sentence from a sequence of phoneme upon which the Viterbi algorithm will be applied, a hidden Markov model and of n-grams. We will also make the theoretical description of techniques for pattern comparaison, neural network analysis and we will attend the creation of a kohonen network from a feature vector.

INTRODUCTION

La reconnaissance vocale est un domaine sujet à la recherche depuis de nombreuses années. Initialement réalisée à l'aide de chaines de Markov et de systèmes probabilistes, les logiciels de reconnaissance vocale ont connu un essor considérable il y a quelques années grâce aux progrès réalisés dans le domaine de l'intelligence artificielle et des réseaux de neurones. En effet, ce genre de logiciel est parfaitement adapté aux techniques du deep learning (apprentissage profond). Ces nouvelles techniques ont permis une amélioration de la qualité des logiciels de reconnaissance vocale. Ces logiciels permettent de simplifier le quotidien en permettant à des machines de faire un travail qu'elles ne peuvent pas faire autrement, comme par exemple l'assistance téléphonique qui peut s'effectuer automatiquement, ou encore la dictée vocale qui permet à l'ordinateur décrire un texte en le dictant.
Le but de ce TIPE sera de réaliser un logiciel de reconnaissance vocale capable de transcrire, sous forme de mots, une phrase prononcée par l'utilisateur. Pour cela nous commencerons par étudier le signal afin qu'il soit utilisable, puis à l'aide d'un modèle acoustique nous transformerons ce signal en une liste de phonème et enfin nous verrons comment retrouver la phrase à partir d'une séquence de phonème.

I- Traitement du signal

1) Réduction du bruit

Déjà, il faut distinguer le bruit produit par le micro (par exemple, un grésillement) du bruit créé par l'environnement.

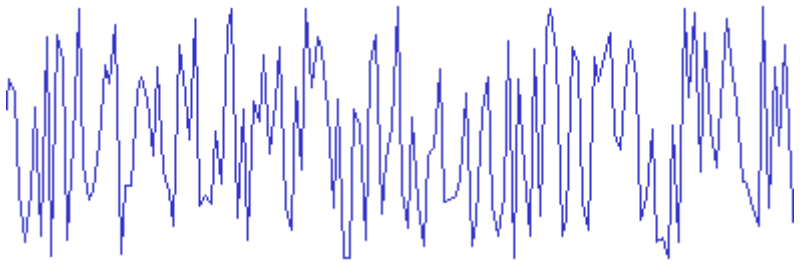

Figure 1: Représentation graphique d'un bruit blanc

Le premier, appelé bruit blanc, est un signal qui mélange toutes les fréquences. Il est donc impossible de l'éliminer grâce à une transformée de Fourier, car dans le domaine fréquentiel, on ne pourra distinguer le bruit du reste. Nous nous sommes donc intéressés à la décomposition en ondelettes (DWT : discret wavelet transform). Celle-ci à la particularité de dépeindre à la fois le comportement temporel et fréquentiel d'une partie du signal. En supprimant tous les éléments de cette décomposition inferieurs à un certain seuil, on peut supprimer le bruit d'un signal. L'implémentation d'une telle décomposition étant complexe, nous avons utilisé la bibliothèque GSL (GNU Science Library). Cependant, les résultats obtenus détérioraient la qualité du signal sans éliminer la totalité du bruit. Nous nous sommes donc tournés vers un logiciel déjà existant donnant de meilleurs résultats.

Nous avons choisi d'utiliser le logiciel SoX. Pour l'utiliser, il faut lui donner un échantillon sonore composé uniquement du bruit généré par le micro afin qu'il établisse un profil du bruit. On peut ensuite utiliser ce profil pour retirer le bruit de tout signal enregistré à l'aide de ce micro.

Le second type de bruit est plus difficile à enlever puisqu'il est extrêmement variable et imprédictible. Aucun effort particulier n'a été fait dans le but de le réduire et la reconnaissance faite par le logiciel sera donc plus efficace dans des environnements relativement calmes.

2) Calcul des vecteurs caractéristiques

a. Fenêtrage et échantillonnage

Le signal sonore que l'on cherche à analyser est le plus souvent non stationnaire. Cependant, celui-ci est constitué d'une suite de phonèmes dont les propriétés ne varient pas pendant une durée pouvant aller de 5 à 100 millisecondes. Ainsi, on supposera que sur une fenêtre de quelques millisecondes, le signal est stationnaire. On utilisera pour cela le fenêtrage de Hamming défini par l'expression suivante :

$$\forall n < N, h(n) = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N-1}\right)$$

La valeur de N étant adaptée pour couvrir 25 ms du signal ($N = 0{,}025 * \nu$ avec $\nu = 16kHz$ la fréquence d'échantillonnage).

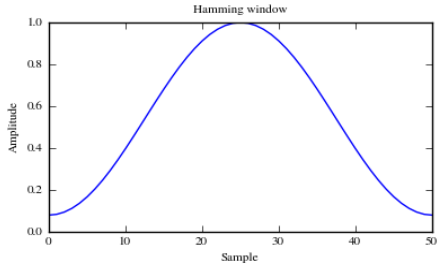

Figure 2: Représentation graphique de la fenêtre

Pour ne pas perdre les informations correspondant aux amplitudes faibles de la fonction de fenêtrage, on créera des fenêtres superposées pendant un laps de temps $Q = 10\ ms$ adapté à la longueur de la fenêtre.

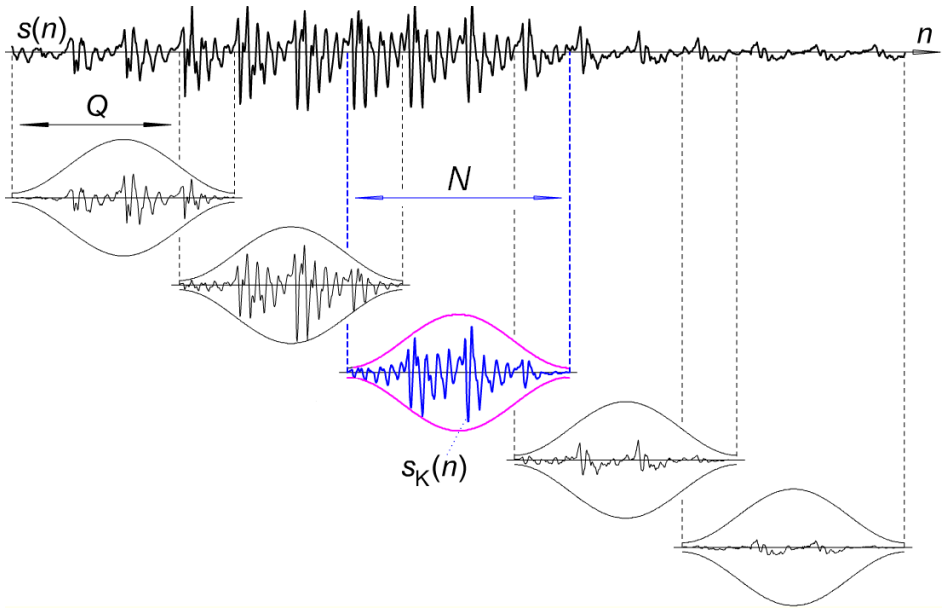

Figure 3: Formation des fenêtres s_k à partir d'un signal échantillonné.

L'expression de la fonction correspondant à la k-ième fenêtre est donc la suivante :

$$\forall n < N, s_k(n) = s(k * Q + n)h(n)$$

Les valeurs de Q et N ainsi que la fonction de fenêtrage sont adaptables. Celles données ci-dessus sont classiques pour effectuer de la reconnaissance vocale et permettent de trouver un bon compromis entre la quantité de calculs et la précision des résultats. La fenêtre de Hamming évite d'importantes discontinuités aux bords de la fenêtre mais pourrait être remplacé par une fenêtre de Hann qui présente des propriétés similaires.

A ce stade, on peut analyser les signaux tronqués sont indépendants. Il faut maintenant extraire le vecteur caractéristique de chaque fenêtre en déterminant ses coefficients.

b. Calcul des MFCCs (Mel Frequency Cepstral Coefficients)

Les premiers coefficients d'un vecteur caractéristique sont appelés MFCC. Des études sur les performances d'un système de reconnaissance vocale en fonction du nombre de MFCC ont montré que 13 coefficients peuvent donner de très bons résultats. Les calculs se font en suivant les étapes présentées ici.

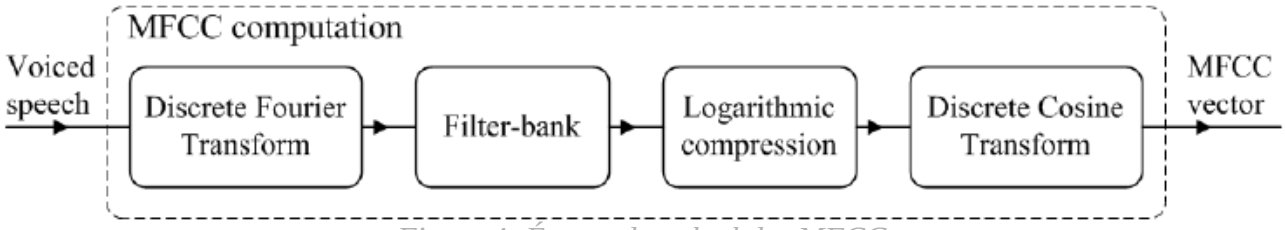

Figure 4: Étapes du calcul des MFCC

Transformée de Fourier

La première étape du calcul des MFCC consiste à calculer la transformée de Fourier S_k du signal tronqué s_k. Ceci est fait grâce à un algorithme de FFT pour économiser du temps de calcul. On complétera ce signal par des zéros de manière à ce qu'il comporte 512 points.

Un signal sonore étant à valeurs réelles, sa transformée est symétrique. On ne considère donc que les M = 257 premiers points.

$$\forall n < N, S_k(N-n) = S_k(n) = S_k(-n)$$

Densité spectrale de puissance et filtres passe bande

Il faut ensuite calculer la densité spectrale de puissance de la fenêtre, définie par :

$$\forall n < \mathrm{M}, P_k(n) = \frac{1}{N}|S_k(n)|^2$$

On peut maintenant filtrer cette fonction avec $N_{filtres} = 23$ filtres passes bandes en échelle mel. Ce filtrage renvoie un nombre réel par filtre qui donne une estimation de la puissance spectrale du signal dans une tranche de fréquence.

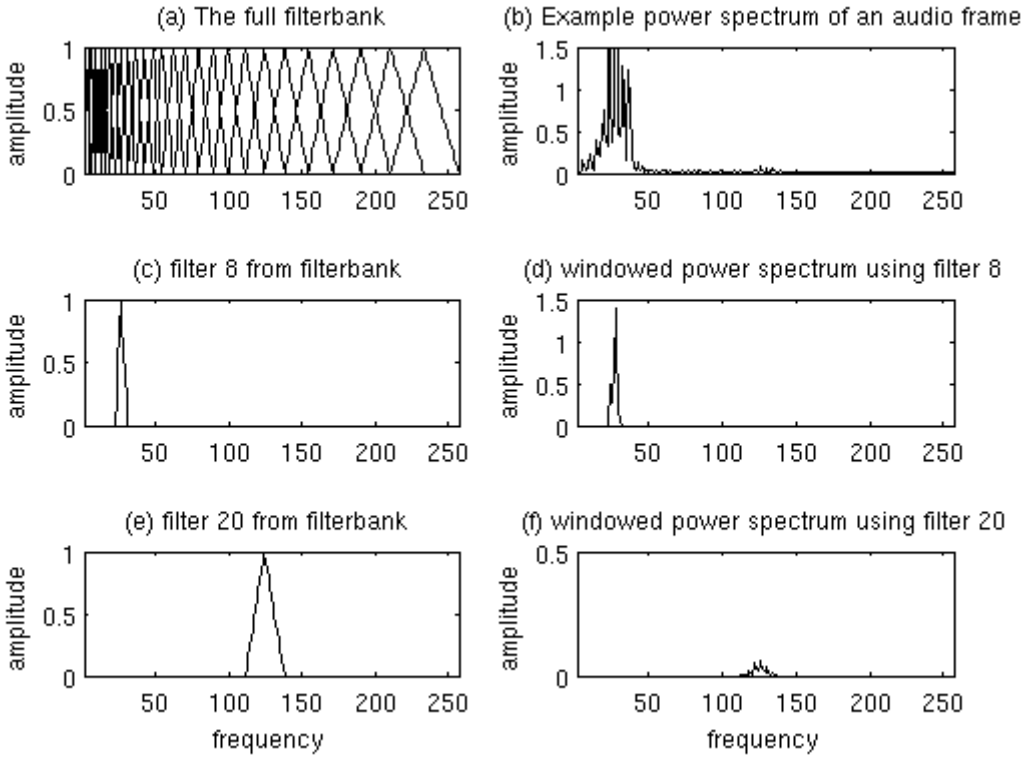

Figure 5: Illustration de l'application des filtres 8 et 20 à un spectre donné

Les filtres sont définis de la manière suivante :

On pose d'abord une fréquence minimum et maximum que l'on passe en échelle mel. On choisit respectivement 0 *Hz* et la fréquence de Nyquist (8 *kHz*), ce qui donne 0 *mel* et 2840 *mel*.

La conversion d'Hertz vers mel se fait de la manière suivante :

$$m = Mel(f) = 1127 * \ln\left(1 + \frac{f}{700}\right)$$

On calcule ensuite la fréquence centrale de chaque filtre en les espaçant linéairement entre 0 et 2840 *mel*. En repassant ces valeurs en Hertz, on obtient des filtres espacés logarithmiquement selon l'échelle des fréquences.

$$\forall m < N_{filtres}, f_m = Mel^{-1}\left(\frac{m}{N_{filtres} - 1} * 2840\right)$$

Pour pouvoir être appliquées au spectre du signal, il faut aligner ces valeurs sur les paramètres de la FFT :

$$\forall m < N_{filtres}, f'_m = round\left(\frac{f_m}{16000} * 512\right)$$

Chaque valeur $f'_m,\ m < N_{filtres}$ représente la fréquence correspondant au pic d'un filtre. On définit le m-ième filtre par :

$$\forall k \in [f'_{m-1}; f'_m], H_m(k) = \frac{k - f'_{m-1}}{f'_m - f'_{m-1}}$$

$$\forall k \in [f'_m; f'_{m+1}], H_m(k) = \frac{f'_{m+1} - k}{f'_{m+1} - f'_m}$$

Une fois filtrées, on obtient les puissances suivantes :

$$\forall m < N_{filtres},, P'_k(m) = \sum_{i=0}^{\frac{N}{2}-1} P_k(i) * H_m(i)$$

Transformée en cosinus discrète (DCT-II)

La dernière étape sert à essayer d'éliminer les caractéristiques dépendant du locuteur en calculant les coefficients cepstraux. Le cepstre d'un signal est défini de la manière suivante :

$$F^{-1}[\log F[s_n]]$$

Le cepstre est donc le spectre d'un spectre. Les harmoniques dépendant du locuteur de la fréquence fondamentale sont donc transformées un coefficient cepstral d'ordre supérieur (figure 6b). La transformation inverse des coefficients les plus bas est montrée en figure 6c et celle des coefficients les plus hauts en 6d. En conservant uniquement les premiers coefficients, on élimine les parasites crées par l'appareil vocal sur le son. Les 13 premiers coefficients forment donc les MFCCs. On n'utilise pas une TFD mais plutôt une transformée en cosinus discrète car celle-ci est plus efficace lorsqu'il s'agit de compresser des données.

$$\forall k < 13, C_k = \sum_{n=0}^{N_{filt}-1} \ln[P'_k(n)] * \cos\left[\frac{k\pi}{N_{filt}}\left(n + \frac{1}{2}\right)\right]$$

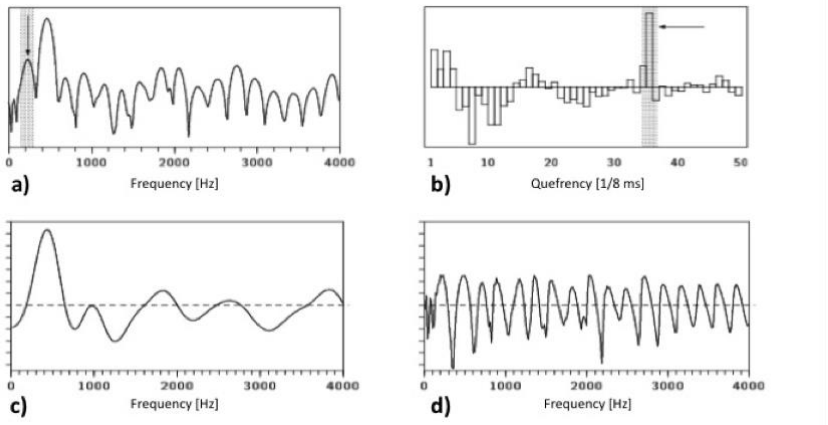

Figure 6: Effet du calcul du cepstre sur la fondamentale

c. Extraction du vecteur caractéristique

Une fois les 13 coefficients spectraux calculés, on calcule leurs dérivées premières et secondes.

$$\Delta C_k = C_{(k+1)} - C_{(k-1)}$$
$$\Delta\Delta C_k = \Delta C_{(k+1)} - \Delta C_{(k-1)}$$

Le vecteur caractéristique est composé de ces trois vecteurs mis bouts à bouts pour un total de 39 coefficients.

Ces opérations ont deux effets primordiaux :

Premièrement, on passe d'une liste de 0.025*16000=400 valeurs par fenêtre à seulement 39. Ceci permet de réduire de façon significative la durée d'apprentissage du réseau de neurones.

D'autre part, les données extraites sont plus représentatives du phonème prononcé et permettent un apprentissage plus simple et des résultats plus précis.

3) Reconnaissance de plusieurs locuteurs

Il existe de deux manières pour créer un système utilisable par plusieurs locuteurs. La première consiste à avoir une base de données contenant suffisamment d'exemple prononcés par une large gamme de voix et la seconde de demander à l'utilisateur de prononcer un ensemble de phrases donné pour calibrer le système, notamment en mesurant les différences avec une prononciation de référence pour établir une proportionnalité entre la voix de l'utilisateur et celle utilisée pour former la base de donnée. La seconde méthode est généralement plus efficace mais demande plus d'effort pour l'utilisateur.

Lorsque l'on calcule un MFCC, le passage au domaine fréquentiel et les filtrages effectués permettent d'éliminer une grande quantité d'éléments dépendant du locuteur, mais l'efficacité n'est pas suffisante. En utilisant la deuxième méthode donnée, on peut récupérer certaines informations sur le timbre vocal de l'utilisateur et ainsi calibrer certains coefficients, notamment les fréquences centrales de la banque de filtre.

Toutefois, les MFCCs se montrent peu efficaces pour reconnaitre des voix aigües. Couplé avec une base de données très maigre, le système développé ne pourra probablement pas reconnaitre beaucoup de locuteurs.

II- Modèle acoustique

Un fois le vecteur caractéristique obtenu, on utilise un modèle acoustique pour détecter le phonème prononcé dans la fenêtre étudiée. Les techniques utilisées à cet effet sont multiples, les principales étant les automates de Markov à états cachés (HMM : Hidden Markov Models) et les réseaux de neurones. Dans cette dernière catégorie, il existe plusieurs variantes que nous présenterons brièvement avant de détailler celle que nous avons choisie.

1) Les réseaux de neurones

Un réseau de neurones artificiels, est un ensemble d'algorithmes dont la conception est à l'origine très schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques.

Il existe une multitude de réseaux de neurones chacun ayant ses avantages et inconvénients. Les plus performants théoriquement sont ceux utilisant les méthodes du deep learning. En effet, dans la première partie, nous avons vu comment calculer les vecteurs caractéristiques d'un fichier audio grâce à une méthode pouvant paraître compliquée à trouver. De plus, il serait nécessaire de créer un autre algorithme si nous voulions classifier d'autres types de données, tel que des images. L'apprentissage profond permet de remplacer ce processus laborieux par un algorithme n'ayant aucune connaissance spécifique du problème étudié. Ces réseaux sont toutefois très complexes et nécessitent beaucoup de temps et de puissance de calcul pour arriver à un résultat.

Une architecture classique de deep learning est celle du Deep Belief Network. Ces réseaux sont formés de couches elles-mêmes constituées de Machine de Boltzmann restreinte que l'on entraine les unes après les autres avant de régler les paramètres plus précisément.

Nous avons préféré les réseaux de neurones aux HMM tout d'abord parce que ils constituent une part importante de la recherche actuelle et qu'ils sont présent dans de plus en plus de logiciels et trouvent des applications dans une large gamme de domaines. Pour la reconnaissance vocale, ils ont complètement remplacé les HMM grâce aux avantages suivants :

- Leur capacité d'entrainement. Les réseaux peuvent apprendre n'importe quelle association d'entrées-sorties à condition qu'il possède une structure appropriée. Ceci est utile pour entrainer le réseau à classifier des échantillons de voix dans des catégories de phonèmes.

- Leur généricité. Les réseaux ne font pas que mémoriser la base de données, ils apprennent leur forme sous-jacente. Ils peuvent donc généraliser ce qu'ils ont appris à de nouveaux exemples. C'est essentiel en reconnaissance vocale car les sons prononcés ne sont jamais exactement les mêmes.

- Leur non-linéarité. Les réseaux peuvent déterminer une régression non linéaire et non paramétrique correspondant à la base de données, ce qui leur permet d'apprendre des motifs en apparence très complexes.

Parmi tous les algorithmes existant pour résoudre le problème, on distingue trois grandes catégories :

- Les algorithmes supervisés. On soumet au réseau une entrée et on compare la réponse à la sortie désirée puis on adapte le réseau de sorte à minimiser l'erreur produite. Un tel algorithme nécessite une base de données pour laquelle on aurait une sortie désirée pour chaque entrée.

- Les algorithmes non supervisés. On soumet au réseau une liste d'entrées et celui-ci se charge de classifier les entrées en un certain nombre de catégories. Pour de la reconnaissance vocale, il faut donner au réseau des échantillons de voix sans indiquer ce qu'ils signifient.

- Les algorithmes semi-supervisés. Ces algorithmes utilisent un ensemble de données étiquetées et non-étiquetés. Il se situe entre les deux autres.

Les réseaux les plus adaptés à la reconnaissance artificielle sont les réseaux de neurones récurrents car ils prennent en compte le comportement temporel des données. Cependant, les algorithmes d'apprentissage sont supervisés et nous voulions un algorithme non supervisé puisqu'il facilitait la création de la base de données. Nous avons donc opté pour les réseaux de Kohonen qui répondait à ce critère.

2) Réseaux de Kohonen

a. Présentation

Les réseaux de Kohonen, ou carte auto adaptative (SOM : Self Organizing Map), initialement conçus pour la reconnaissance vocale, permet de réaliser des tâches de réduction de dimension et de classification. Nous les utiliserons pour classifier les vecteurs caractéristiques selon des critères sélectionnés dans la base de données lors de la phase d'apprentissage

Un tel réseau se présente sous la forme d'un tableau à 2 dimensions dont chacune des cases contient un vecteur à 26 dimensions. Une case est appelée neurone et le vecteur qui lui correspond est son poids.

Après l'apprentissage, cette carte fournira une représentation discrète de l'espace d'entrée, permettant d'indexer des données via leurs coordonnées dans le tableau. La structure topologique de l'espace de départ est conservée lors de la réduction de dimension.

b. Algorithme d'apprentissage

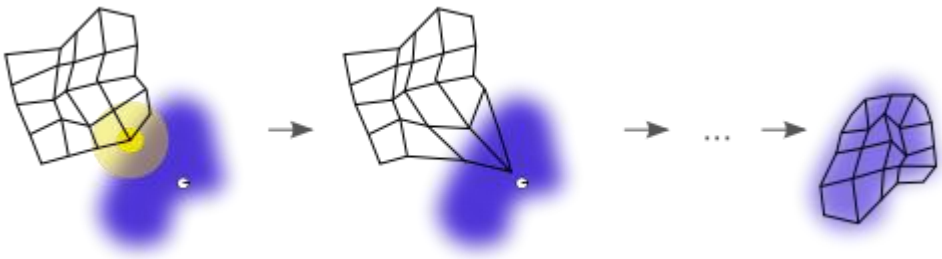

Figure 7: Apprentissage d'un SOM

Sur la figure 6, on peut voir une illustration de l'apprentissage d'un SOM de $\mathbb{R}^2$ dans $\mathbb{R}^2$. La forme bleue représente les données et le disque blanc est le vecteur en train d'être traité. Le poids du neurone gagnant ainsi que celui de son voisinage (en jaune) est adapté. Au fur et à mesure de l'avancée de l'algorithme, la carte classifie de mieux en mieux l'ensemble des données.

Ce réseau est donc fondé sur une méthode d'apprentissage non supervisé. Il se distingue par l'utilisation d'une fonction de voisinage qui conserve la topologie de l'espace de départ et par un apprentissage compétitif, en opposition à un apprentissage par correction de l'erreur: ici, les neurones répondant à une entrée seront « récompensés ».

L'initialisation des valeurs de poids pour chaque neurone se fait aléatoirement. Ensuite, l'algorithme est itératif :

A chaque itération, on soumet un vecteur choisi aléatoirement dans la base de données à la carte auto adaptative. Selon les valeurs des poids des neurones, il en sortira un neurone gagnant, appelé BMU (Best Matching Unit). On adapte alors tous les neurones présents dans le voisinage du BMU, en rapprochant leur poids du vecteur d'entrée. Ceci leur permettra de répondre encore mieux à d'autres entrées semblables à celle-ci. Ainsi, c'est toute la région de la carte autour du neurone gagnant qui se spécialise.

A chaque itération, le voisinage d'un neurone se rétrécit et la modification des poids devient de moins en moins importante.

Formalisation mathématique

t et t_{max} désignent l'itération actuelle et le nombre d'itérations.
N désigne le nombre de neurones dans la grille.
w_n désigne le poids du $n^{ième}$ neurone dans la grille.
p_n désigne la position du $n^{ième}$ neurone sur la grille.
$\|.\|$ désigne la norme euclidienne.

1. On donne une valeur aléatoire aux poids de chaque vecteur.

2. On sélectionne un vecteur d'entrée V.

3. On note s un neurone tel que $\forall n \leq N, \|w_s(t) - V\| \leq \|w_n(t) - V\|$. s est le BMU.

4. On adapte les poids des neurones dans le voisinage du BMU selon la formule suivante :

$$\forall n \leq N, w_n(t+1) = w_n(t) + \mathrm{h}(n, s, t) * \alpha(t) * (V - w_n(t))$$

Avec $h(n, s, t) = \exp\left(-\frac{\|p_n - p_s\|}{2 * \theta(t)^2}\right)$,

$\theta(t) = r_a * \left(\frac{r_b}{r_a}\right)^{\frac{t}{t_{max}-1}}$ donnant le rayon du voisinage d'un neurone à l'étape t

et $\alpha(t) = l_a * \left(\frac{l_b}{l_a}\right)^{\frac{t}{t_{max}-1}}$ donnant le coefficient d'apprentissage à l'étape t.

5. On incrémente t et on recommence depuis l'étape 2 tant que $t < t_{max}$.

Une variante de l'algorithme consiste à parcourir tous les vecteurs de la base de données à chaque itération.

c. Paramètres variables

Le choix des paramètres de l'algorithme est très important. On en compte 7 :

- Les dimensions de la carte
- Le nombre d'itérations
- Les deux valeurs extrêmes du rayon
- Les deux valeurs extrêmes du coefficient d'apprentissage

De plus, l'apprentissage se déroule souvent en deux fois. Une première fois avec peu d'itérations et un rayon et un coefficient élevé pour rapprocher la carte de l'ensemble des données. Puis une seconde fois de manière plus fine et avec un nombre d'itérations beaucoup plus élevé.

d. Représentation graphique

Pour pouvoir visualiser les résultats du réseau, on calcule généralement l'U-Matrice du réseau. Cela consiste à calculer la distance moyenne séparant le poids d'un neurone de ceux de ses voisins puis de l'afficher sur une carte. Plus cette distance sera élevée, plus le neurone sera représenté avec une couleur foncée.

Une telle représentation permet de rapidement visualiser les groupes formés pour ensuite les classer.

Cependant, lorsque les délimitations entre les groupes sont moins marquées, il est parfois impossible d'attribuer à un neurone de la carte une seule catégorie. On peut donc créer une troisième carte sur laquelle il y aura les catégories possibles auxquelles peut appartenir un neurone.

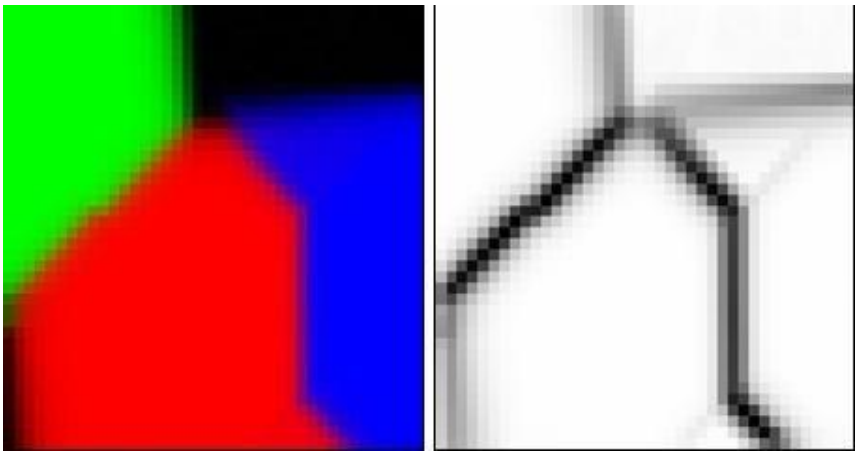

Figure 8: Un SOM classant des vecteurs de $\mathbb{R}^3$. A gauche, les poids représentés en format RGB et à droite, l'U-Matrice du réseau.

III- Reconstitution de phrase

Une fois le signal transformé en une liste de phonème par le réseau de neurone, il faut transformer ces phonèmes en mots.

1. Modèle de Markov caché (MMC)

Pour comprendre le modèle de Markov caché il faut d'abord comprendre les chaines de Markov. Une chaine de Markov est constituée de plusieurs états ainsi que d'une probabilité de passage d'un état à un autre. On peut ainsi, depuis n'importe quel état regarder la probabilité de passage à tous les autres états.
Dans une chaine de markov la probabilité d'être dans un état ne dépend que de l'état précédent : c'est le modèle de Markov.

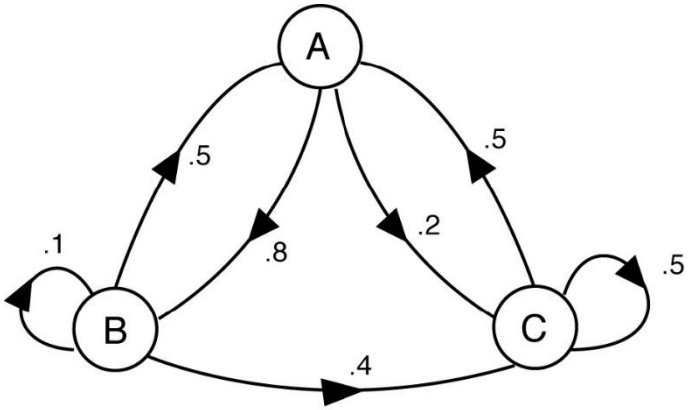

Figure 9: Représentation d'une chaine de Markov contenant trois états

Dans un modèle de Markov caché, certains des états ne sont pas visible, il est néanmoins possible de retrouver ces états cachés en observant ce qui est visible. Les états cachés sont la cause des états observables, chaque état caché aura donc une somme de probabilité de cause égal à 1.

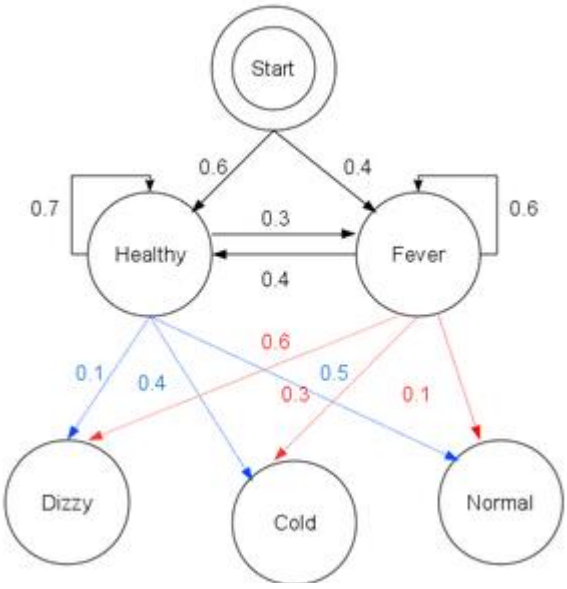

Figure 10: représentation d'une chaine de Markov cachée

Dans la chaine de Markov présente en figure 10, l'on ne peut voir que les états Dizzy, Cold ou Normal. En suivant l'évolution de ces trois symptômes, on peut déterminer si la personne est dans l'état Fever ou dans l'état Healthy.
On étudie donc l'évolution de la transition des états observables pour deviner l'évolution de la transition des états cachés.

Ce modèle de Markov caché est applicable à la reconstitution de phrase : en prenant les phonèmes en sortie du réseau de neurone comme état observable et les mots existant en état caché il est possible, à partir de la transition des phonèmes observable de retrouver le mot originel. Le type de modèle de Markov utilisé est souvent un « left to right hidden Markov model » qui a pour particularité de n'aller que dans un seul sens, avec une boucle pour éviter qu'un phonème qui se répète en sortie du réseau de neurone ne soit présent plusieurs fois.

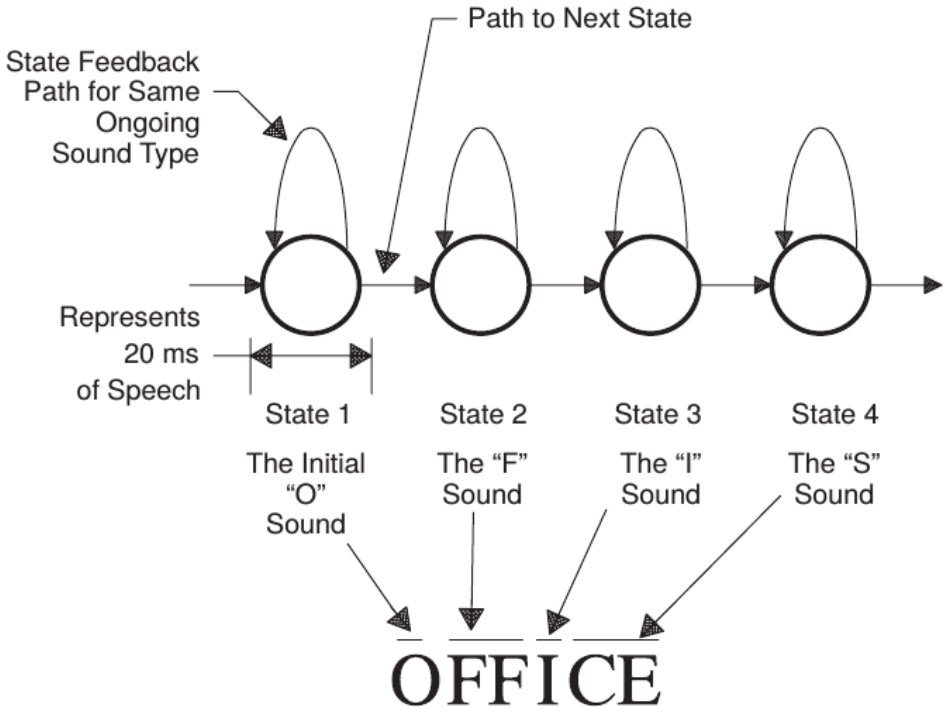

Figure 10 : représentation d'une left to right hidden Markov model qui retrouve le mot office à partir de ses phonèmes

Notre base de données étant triée il est également possible de retrouver un mot en regardant chaque phonème qui lui appartient. Cette méthode bien que plus simple à mettre en œuvre possède un grand défaut : en cas d'ajout, d'erreur ou de retrait d'un phonème il sera impossible de retrouver le mot.
C'est pour cela que les left to right hidden Markov model sont bien supérieure, elles peuvent procurer une probabilité pour chaque mot permettant de retrouver un mot même en présence d'erreur dans la séquence de phonème.

Après avoir pu identifier un mot, le programme n'a pas terminé, il faut que la phrase ait du sens, pour cela on va utiliser les n-gramme. Il a beaucoup de mots qui sont identiques phonétiquement, de plus cette méthode permettra d'éliminer d'avantage d'erreur tel que des mots qui n'aurait pas dû être trouvé.

2. Algorithme de Viterbi

Afin d'obtenir le résultat le plus précis possible, il est plus intéressant de chercher quelle série de phonème est la plus probable. Pour cela l'algorithme de Viterbi est un moyen de calculer cette séquence la plus probable avec une complexité nettement inférieur comparée à tester toutes les listes de phonèmes possible et renvoyer la plus probable.
Pour cela on va, pour chaque états, regarder la probabilité de passer d'un état à un autre ainsi que la probabilité de l'état suivant et répéter l'action jusqu'à parcourir toute la séquence de phonème. Ainsi le nombre de calculer sera grandement réduit.

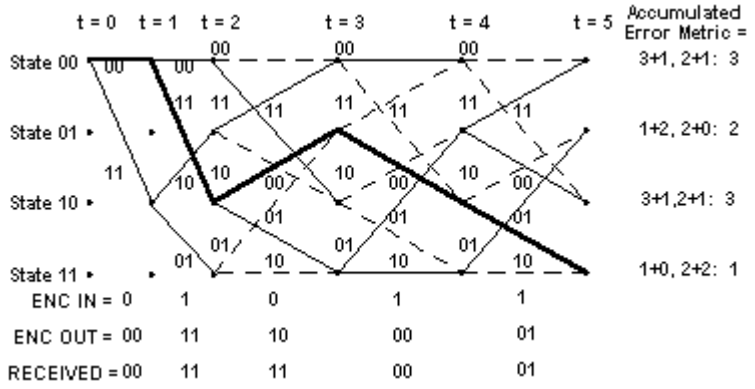

Figure 11: Représentation d'une application de l'algorithme de Viterbi qui trouve le chemin le plus probable

Cette méthode fonctionne de la même manière qu'un réseau de Markov caché.

3. N-gramme

Un n-gramme est une sous séquence de n éléments construit à partir d'une séquence donnée. Un 1-gramme est appelé un unigramme, un 2-gramme est appelé un bigramme et un 3-gramme est appelée un trigramme. Les n-grammes permettent de calculer la probabilité d'apparition d'un mot connaissant les n mots le précédant. Pour créer un n-gramme, il suffit donc en théorie de regrouper et toutes les séquences de n mots apparaissant dans un recueil de texte.
Le n-gramme que l'on utilise est celui crée par Google en collaboration avec l'équipe de Google books. En effet, cette dernière a numérisé un grand nombre de livres en français écrits entre 1500 et 2008. Les textes ont donc été utilisés pour générer des n-grammes, pour n allant de 1 à 5.

Cela représente énormément de données auxquelles on peut accéder via leur site (https://books.google.com/ngrams). Notre programme accède à ce site et requiert donc une connexion internet pour fonctionner.

Un n-gramme correspond à une chaine de Markov : on regarde la probabilité d'apparition d'un mot en fonction des n mots précédents.

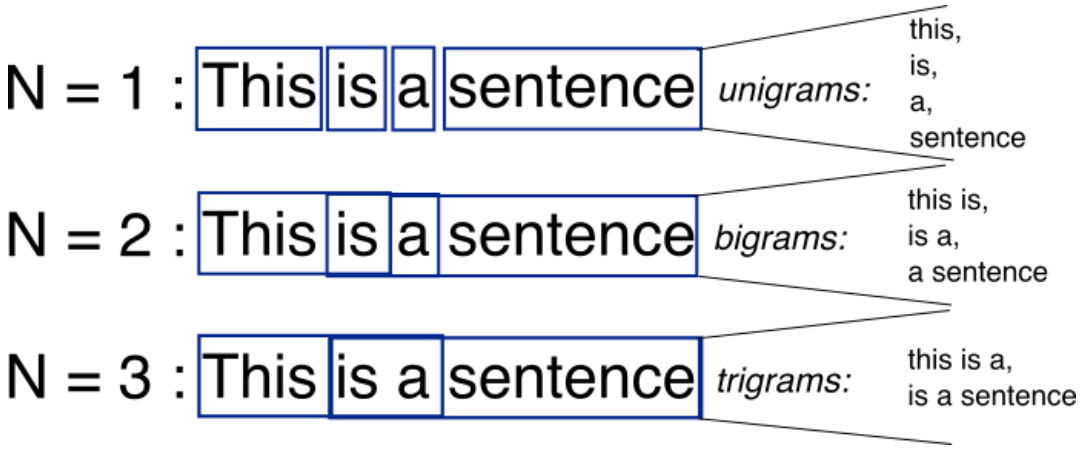

Figure 12: exemple d'unigramme, bigramme et trigramme

Pour un n-gramme si on note w une phrase, w_i le ième mot de cette phrase et hi les n mots précédent w_i, on a alors :

$$P_n(w) = \prod_{i=1}^{n} P(w_i|h_i)$$

Avec $P_n(w)$ la probabilité

P_1 (w_1, w_2, w3, w_4) = P ("this") * P ("is ") * P (« "a" ») * P (« "sentence" »)
P_2 (w_1, w_2, w3, w4) = P ("this"|<s>) * P ("is"|"this") * P ("a"|"is") * P ("sentence" |"a")
P_3 (w_1, w_2, w3, w4) = P ("this"|<s> <s>) * P ("is"|<s>,"this") * P ("a"|"this","is") * P ("sentence" |"is","a")
(<s> représente une absence de mot)

Ce qui donne donc :

$$P_n\ (w)\ =\ \prod_{i=1}^{n} P\ \ (w_i|h_i)$$

Cependant les probabilités des n-grammes sont très proches de 0 pour les 4 ou 5-grammes, pour éviter des problèmes on additionne les logarithmes des probabilités au lieu de les multiplier.
Même si ces probabilités sont infimes elles possèdent un rôle important pour la reconstitution de mot puisqu'on suppose que la phrase énoncé par le locuteur à un sens et les n-gramme sont là pour le retrouver.

On peut remarquer l'importance d'avoir n le plus grand possible en regardant les phrases générées en figure 14 (en anglais). La méthode suivie est de commencer avec n mots puis de rajouter le mot le plus probable déterminé grâce au n-gramme. On remarque que plus il y a de mots pris en compte,

plus la phrase à un sens. Bien évidemment, le n-gramme grandit de manière significative quand n augmente.

Pour finalement obtenir la probabilité de la phrase, on peut additionner les probabilités de la phrase pour chaque n-gram multipliée par un coefficient choisit, ce qui donne

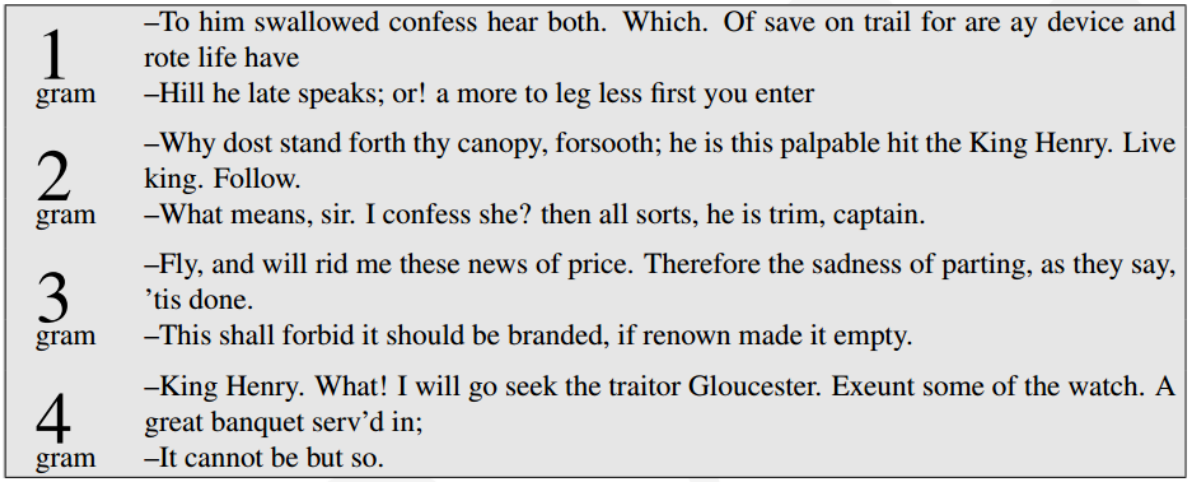

Figure 13: phrase générés en utilisant des n-grams

$$P(w) = \sum_{i=1}^{5} a_i * P_i(w)$$

Avec $a_1 + a_2 + a_3 + a_4 + a_5 = 1$
De cette manière on peut augmenter où réduire l'impact des différentes n-gram afin d'améliorer le résultat.

La probabilité des 5-grammes étant extrêmement basse, augmenter a_5 devrait donc donner en général des résultats plus cohérents au niveau de la phrase, dans le cas contraire la probabilité de p5 est tellement basse qu'on pourrait les négliger ce qui nuirait à la précision du système.

Pour déterminer quelle phrase sera retenue au final il faudra agir de la même manière, en prenant en compte les probabilités obtenues en cours des n-gram ainsi que les probabilités obtenus avec les chaines de Markov afin de déterminer quelle phrase est la bonne.

IV- Configuration et expériences

Le système total possède un nombre important de paramètres pour la plus part définis grâce des expérimentations. Les valeurs choisies pour le calcul des MFCCs ont toutes étés données en partie I et sont des valeurs classiques utilisées par un grand nombre de logiciels de reconnaissance vocal.

Pour la première phase de test, nous avons essayé de reconnaitre une base de données contenant la prononciation des chiffres de zéro à trois. Cela représente 10 phonèmes différents rendant l'apprentissage du réseau assez rapide et nous permettant de réaliser de nombreux tests avec des paramètres différents et ainsi voir leur influence sur les résultats obtenus.

Nous sommes ensuite passés à une base de données constituée de deux phrases contenant chacune la totalité des phonèmes du français. Une fois l'apprentissage fait, la reconnaissance était efficace pour les voyelles mais peu pour les consonnes. Nous en avons déduit que le problème venait en partie de la base de données. En effet, les voyelles étaient naturellement très présentes dans les phrases choisies tandis que certaines consonnes n'apparaissaient qu'une seule fois (les phonèmes peuvent être classés en voyelles et consonnes, voir en annexe).
Nous avons donc changé de base de données pour en choisir une qui contienne chaque couple de consonne puis voyelle possible. Celle-ci regroupe donc tous les phonèmes choisis de manière plus ou moins égale.
Les phonèmes choisis sont au nombre de 30. En effet, bien qu'il existe 36 phonèmes en français, la différence entre certains est très fine voire même inexistante chez certains locuteurs. De plus, nous avons choisi de rajouter un phonème représentant un silence. La liste est donnée en annexe.

Le réseau de Kohonen sera une carte de 21×17 vecteurs de $\mathbb{R}^{39}$.
Les fonctions donnant le coefficient d'apprentissage et le rayon en fonction de l'étape sont celles définies en II-3)b. avec les paramètres suivants.

Variable	Première phase	Seconde phase
t_{max}	150	10000
l_a	0.5	0.3
l_b	0.1	0.001
r_a	13.6	6.8
r_b	6.8	0.001

Tableau 1: Valeurs des paramètres pour chacune des phases d'apprentissage

V- Annexes

1. Liste des phonèmes utilisés :

Phonème utilisé - type	Phonème(s) remplacés en notation internationale (API)	Exemple de mot contenant le phonème
I - voyelle	/i/	**l**it, **l**yre
ET - voyelle	/e/	**é**cole, bl**é**
AI - voyelle	/ɛ/	**Ai**gle, m**e**rci
A - voyelle	/a/ /ɑ/	**a**vion, p**â**te
AU - voyelle	/ɔ/	**o**s, f**o**rt
O - voyelle	/o/	m**o**t, **eau**
OU - voyelle	/u/	**Ou**rs, ge**nou**
U - voyelle	/y/	Tort**u**e, **r**ue
E - voyelle	/ø/ /ə/	**Re**quin, f**eu**
EU - voyelle	/œ/	M**eu**ble, p**eur**
AN - voyelle	/ɑ̃/	**An**ge, s**an**s
ON - voyelle	/ɔ̃/	L**on**, **o**mbre
IN - voyelle	/ɛ̃/ /œ̃/	Lap**in**, l**un**di
P - consonne	/p/	**P**omme, sou**p**e
T - consonne	/t/	**T**ambour, **t**erre
K - consonne	/k/	**C**adeau, s**ac**
B - consonne	/b/	**B**alle, ro**b**e
D - consonne	/d/	**D**ent, **d**ans
G - consonne	/g/	**G**âteau, ba**g**ue
F - consonne	/f/	**F**usée, **ph**oto
S - consonne	/s/	**S**erpent, ta**ss**e
CH - consonne	/ʃ/	**Ch**at, tâ**ch**e
V - consonne	/v/	**V**ache, **v**ous
Z - consonne	/z/	**Z**èbre, **z**éro
J - consonne	/ʒ/	**J**upe, **g**ilet
L - consonne	/l/	**L**une,**l**ent
R - consonne	/ʁ/	**r**ue
M - consonne	/m/	**M**outon, **m**ot
N - consonne	/n/ /ɲ/	Pei**gn**e, **N**ain

2. Extraits de code en c++

```
void MFCCComputer::buildFilterBank(double _lower, double _upper)
{
    filterBank.resize(Nfilt, vector<double>(K, 0) );

    vector<double> m(Nfilt+2);

    auto mel  = [] (double _f) { return 1127.0 * log(1.0 + _f/700.0); };
    auto iMel = [] (double _m) { return 700.0 * (exp(_m/1127.0) - 1.0); };

    double lower = mel(_lower);
    double upper = mel(_upper);

    double iS = 1.0 / (m.size()-1);
    for (unsigned i(0) ; i < m.size() ; i++)
    {
        double t = i * iS;
        m[i] = (1.0-t) * lower + t * upper;
        m[i] = iMel(m[i]);
        m[i] = round( 512.0 * m[i] / 16000.0);
    }

    for (unsigned i(1) ; i < m.size()-1 ; i++)
    {
        for (unsigned k(m[i-1]) ; k < m[i]   ; k++)
            filterBank[i-1][k] = (k - m[i-1] + 1) / (m[i] - m[i-1] + 1);

        for (unsigned k(m[i])   ; k <= m[i+1] ; k++)
            filterBank[i-1][k] = (m[i+1] - k + 1) / (m[i+1] - m[i] + 1);
    }
}
```

```
void MFCCComputer::buildHammingWindow()
{
    double coef = 2.0*PI / (double)(Nlength-1);
    hamming.resize(Nlength);

    for(unsigned i(0) ; i < Nlength ; i++)
        hamming[i] = 0.54-0.46*cos(coef * i);
}
```

```
void MFCCComputer::computeMFCC(Vector& _result, unsigned _frame)
{
    const std::vector<double>& s = signal.data;

    _result.data.resize(coefCount);

    const unsigned firstPoint = Nstep * _frame;
    const unsigned sup = min(Nlength, s.size() - firstPoint);

    double data[Nfft];
    double fft[2*Nfft];


    /// Fenetrage de Hamming
        for (unsigned i(0) ; i < sup ; i++)
            data[i] = s[firstPoint+i] * hamming[i];

    /// Zero padding
        for (unsigned i(sup) ; i < Nfft ; i++)
            data[i] = 0.0;

    /// FFT de longueur 512
        for (unsigned i(0) ; i < Nfft ; i++)
        {
            fft[2*i+0] = data[i];
            fft[2*i+1] = 0.0;
        }

        gsl_fft_complex_radix2_forward (fft, 1, Nfft);

    /// Densite spectrale de puissance
        for (unsigned k(0) ; k < K ; k++)
            data[k] = (fft[2*k]*fft[2*k] + fft[2*k+1]*fft[2*k+1]) * iNfft;

    /// Application de la banque de filtres
        vector<double> logFilterOutput(Nfilt, 0);

        for (unsigned i(0) ; i < filterBank.size() ; i++) // [0, Nfilt[
        {
            for (unsigned j(0) ; j < filterBank[i].size() ; j++) // [0, K[
                logFilterOutput[i] += filterBank[i][j] * data[j];

            logFilterOutput[i] = logFilterOutput[i] < 2e-22 ? -50.0 :
log(logFilterOutput[i]);
        }

    /// Calcul de la DCT
        for (unsigned k(0) ; k < coefCount ; k++)
        {
            for (unsigned n(0) ; n < Nfilt ; n++)
                _result[k] += logFilterOutput[n] * cos(PI * k * ((double)n +
0.5) * iNfilt);
        }

    /// Energie
        if (useLogEnergy)
        {
            double energy = 0.0;

            for (unsigned i(0) ; i < sup ; i++)
```

```
                energy += s[firstPoint+i] * s[firstPoint+i];

            _result[0] = energy < 2e-22 ? -50.0 : log(energy);
        }
}



bool SOM::epoch()
{
    if (getTrained())
        return false;

    double sRad = rad * rad;
    double sig = - 1.0 / (2.0*sRad);


    random_shuffle(db.permutation.begin(), db.permutation.end());

    for (unsigned t: db.permutation)
    {
        Node BMU = getBMU(db[t]);

        for (unsigned i(0) ; i < w ; i++)
        {
            for (unsigned j(0) ; j < h ; j++)
            {
                unsigned a = BMU.first - i, b = BMU.second - j;
                double l2 = a*a + b*b;

                Vector& NWeight = grid[i][j];

                NWeight += lr * exp(l2 * sig) * ( db[t] - NWeight );
            }
        }
    }


    if (++db.step == db.maxSteps)
    {
        db.step = 0;
        return false;
    }

    lr *= db.dL;
    rad *= db.dR;

    return true;
}
```

CONCLUSION

Nous avons décrit une des nombreuses manières d'effectuer la reconnaissance automatique de la parole. L'importance d'avoir une reconnaissance automatique de la parole précise se fait de plus en plus ressentie. Cependant ce type de logiciel présente de nombreuse contrainte au niveau du vocabulaire, de l'indépendance du locuteur et de manière continue. De plus l'environnement a un impact important sur la qualité de la reconnaissance. En utilisant des réseaux de neurones plus puissant tel que le deep belief network il est possible de créer une transcription très précise indépendante du locuteur tout en pouvant transcrire la parole de manière continue, cependant la base de donnée nécessaire pour entrainer de tel réseau est extrêmement large et nécessite des milliers d'heures. Maintenant que nous arrivons a faire des systèmes performants même sur des téléphone portable grâce a internet. Mais l'entrainement de tel reconnaissance automatique de la parole reste compliqué, il dure souvent plusieurs jours. Les progrès en reconnaissance vocale ont beaucoup avancé ces dernières années mais le sujet est encore d'actualités puisque la théorie derrière le sujet est relativement compliquée. Il y a encore de nombreux point qu'on pourrait améliorer et pour lesquels la recherche doit progresser d'avantage avant d'arriver a un logiciel de reconnaissance vocale qui pourrait rivaliser avec la capacité du cerveau humain.

SOURCES :

http://www.etsi.org/deliver/etsi_es/201100_201199/201108/01.01.03_60/es_201108v010103p.pdf
http://www.cs.toronto.edu/~graves/phd.pdf
http://isl.anthropomatik.kit.edu/cmu-kit/downloads/PhD_1995_Tebelskis.pdf
http://www.k-netweb.net/projects/tipe/reseauxdeneuronesartificiels.pdf
http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html
http://cache.media.eduscol.education.fr/file/ecole/36/3/mots_nature_frequence_124997_292363.pdf
www.recognize-speech.com (site hors service mais accessible depuis
http://web.archive.org/web/20161024183012/http://recognize-speech.com:80/)
http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
http://fr.wikipedia.org/wiki/Carte_auto_adaptative
https://en.wikipedia.org/wiki/Self-organizing_map
https://en.wikipedia.org/wiki/Viterbi_algorithm
https://fr.wikipedia.org/wiki/Phon%C3%A8me
http://www.xavieranguera.com/tdp_2011/10-HMM.pdf
http://web.science.mq.edu.au/~cassidy/comp449/html/ch12s02.html
https://lagunita.stanford.edu/c4x/Engineering/CS-224N/asset/slp4.pdf
https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf