Projet de fin d'études

ENSEEIHT

par

Emmanuel Ravelli

PRÉDICTION NON-LINÉAIRE POUR LE CODAGE AUDIO SANS PERTES

effectué au groupe de recherche sur la parole et l'audio
de l'université de Sherbrooke
sous la tutelle de :

Philippe Gournay

Remerciements

Je tiens tout particulièrement à remercier Monsieur Philippe Gournay, qui a encadré ce projet de fin d'étude avec enthousiasme, et a su me conseiller efficacement tout en me laissant travailler très librement.

Merci également à :

- Monsieur Roch Lefebvre, pour m'avoir acceuilli au sein de son groupe de recherche.
- Monsieur Jean-Yves Tourneret (enseignant-chercheur à l'ENSEEIHT), sans qui ce stage n'aurait pas eu lieu.
- Madame Corinne Mailhes (enseignant-chercheur à l'ENSEEIHT), pour son enseignement en compression du signal.
- Mes parents, pour l'aide qu'ils m'ont apporté durant ce stage.
- Mes amis, pour leur soutien moral.

Table des matières

Introduction

De nos jours, de plus en plus de médias sont numérisés (image, son, vidéo...) et stockés ou transmis dans un format numérique. Cela a plusieurs avantages, dont le plus important est l'absence de perte de qualité du média numérisé pendant son transport ou lors de traitements sur celui-ci (à l'aide de logiciels, par exemple). Il existe également quelques inconvénients intrinsèques au caractère numérique, notamment la perte de précision lors du passage du monde analogique au monde numérique (appelé conversion analogique / numérique ou échantillonnage). Pour que le signal numérisé soit rigoureusement le même que le signal d'origine, il faudrait numériser ce dernier avec une précision infinie, donc en stockant une quantité infinie d'informations... C'est évidemment très théorique et impossible à réaliser. En pratique, la limite de précision nécessaire est fixée généralement par celle du récepteur final de ce média : l'œil pour l'image, l'oreille pour le son. Or, pour des utilisations professionnelles, cette limite peut être fixée assez haute (96khz/24bits pour le son) et les fichiers numériques prennent alors rapidement beaucoup de place : 2.304 Mbits pour une seconde d'une piste mono (à 96khz/24bits). Comme les studios enregistrent généralement leurs travaux en multi-pistes, l'espace nécessaire au stockage devient très important. C'est pourquoi il a été nécessaire de développer des algorithmes pour réduire la taille de ces fichiers audio : c'est la compression audio (ou codage audio).

Les algorithmes de compression audio les plus connus (MP3, WMA, OGG...) sont des algorithmes de compression avec perte (ou compression destructive) car la compression enlève des informations du fichier audio : le fichier original et le fichier décompressé ne sont pas identiques. Or, ce type de compression n'est pas adapté au travaux réalisés dans les studios. En effet, pourquoi faire beaucoup d'effort à travailler le mixage d'un morceau en finesse, l'échantillonner en 24 bits / 96 KHz, faire attention à la qualité du son dans les moindres détails, si c'est pour finalement utiliser une compression destructive et ainsi perdre tous les détails du morceau ? C'est pourquoi il a été nécessaire de développer des algorithmes de compression sans perte.

Depuis un an, le groupe de recherche sur la parole et l'audio de l'université de Sherbrooke acceuille des étudiants en stage afin de travailler sur le codage audio sans perte. Le premier stage a consisté en l'étude, l'implémentation et l'optimisation d'un algorithme de compression audio sans perte. L'algorithme développé utilisait une technique de prédiction linéaire mais n'exploitait pas

les similitudes qui existent entre les deux canaux d'un signal stéréo. Le second stage s'est donc naturellement focalisé sur l'aspect stéréo du codage audio sans perte. L'algorithme qui y a été développé utilisait une technique de prédiction linéaire stéréo. Cependant, la prédiction linéaire présente des limites. C'est pourquoi le stage que j'ai effectué et qui est une suite logique des deux stages précédents, porte sur un autre type de prédiction : la prédiction non linéaire.

Le premier chapitre de cette étude présente quelques généralités sur le codage audio sans perte. Les deuxième et troisième parties se consacrent à l'étude de deux types de prédiction non linéaire : respectivement la prédiction avec un filtre de Volterra et la prédiction avec un réseau de neurones. On a ensuite retenu la technique qui donnait les meilleurs résultats et on l'a implementée en langage C, dont les spécifications sont décrites dans le chapitre 4. Enfin le dernier chapitre ouvre la présente étude sur de nouvelles voies qui pourraient être explorées en codage audio sans perte.

Chapitre 1

Le codage audio sans perte

1.1 Codage audio avec perte et codage audio sans perte

Le codage audio (ou compression audio) permet de réduire la taille d'un fichier audio pour qu'il tienne le moins de place possible sur le disque dur, ou qu'il soit plus facilement exportable via disquette ou réseau. Pour ce faire, il faut réduire le nombre de données contenues dans le fichier audio, soit en optimisant la façon dont ce dernier inventorie les données, soit en supprimant les informations qui s'avèrent superflues. Suivant qu'on fait appel à l'une ou l'autre de ces méthodes, on distinguera 2 types de codage : le codage avec perte et le codage sans perte.

Le codage audio avec perte permet d'obtenir des ratios de compression très élevés (de l'ordre de 12 pour le mp3) mais pour cela, le codage audio avec perte doit supprimer des données : le fichier décompressé n'est pas identique au fichier original. Cependant, à l'écoute, on distinguera difficilement les 2 fichiers. Le codage audio avec perte est donc destiné uniquement à l'écoute de fichiers audio. Il existe beaucoup d'algorithmes différents de codage audio avec perte suivant l'utilisation qui en est faite : les algorithmes destinés à la transmission de données en streaming par internet (Real Audio G2, Microsoft ASF), les algorithmes dédiés au codage de la parole (ADPCM, CELP, ACELP...), les algorithmes destinés au codage de fichiers audio (MP3, Twin VQ, VQF, ATRAC...).

Le codage audio sans perte donne des ratios de compression beaucoup moins élevés (de l'ordre de 2 ou 3). Mais, comme son nom l'indique, avec le codage audio sans perte, il n'y a aucune perte de données, c'est à dire que le fichier original et le fichier décompressé sont strictement identiques. Le codage audio sans perte est donc plutot destiné à l'archivage de fichiers audio en vue d'un éventuel travail sur le fichier original (par exemple pour le mastering en studio). Le nombre d'algorithmes de codage audio sans perte est assez limité : il existe beaucoup de codecs (Monkey Audio, LTAC, LPAC, Shorten, WavZip, MUSICompress...) mais les techniques utilisées sont très semblables ; seul LTAC se distingue du lot car il utilise une technique très différente de tous les autres codecs.

1.2 Structure d'un codeur audio sans perte

Le schéma de base pour un codeur audio sans perte dans le cas simple d'un signal monophonique est le suivant [2] :

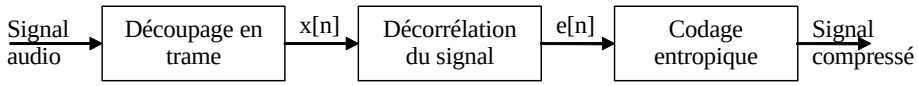


Fig. 1.1 - Schéma bloc d'un codeur audio sans perte

Le signal audio est dans un premier temps décomposé en trames consécutives. La deuxième étape consiste à enlever la redondance d'une trame du signal audio : c'est la décorrélation du signal. Il y a deux grandes familles de codeur audio sans perte qui se distinguent par la méthode qu'ils emploient pour décorréler le signal :

- Les codeurs qui utilisent la prédiction linéaire
- Les codeurs qui utilisent une transformée orthogonale

Dans le cas simple de la fig. 1.1, seul la décorrélation intra-canal est représentée. De nombreux codeurs se contentent de coder séparément les canaux d'un signal audio multi-canaux comme s'il s'agissait de signaux monophoniques. Mais si on veut profiter des similitudes qui existent entre les différents canaux, il faudra envisager une décorrélation inter-canaux. On reviendra sur ce type de décorrélation plus tard.

Le signal obtenu après décorrélation, qu'on appelle résidu, est codé à l'aide d'un codage entropique : c'est la dernière étape. Le fichier compressé est alors composé :

- du résidu
- des paramètres de la prédiction linéaire ou de la transformée orthogonale.

Le résidu a une amplitude moyenne qui est bien inférieure à celle du signal original et une densité spectrale relativement plate. Une amplitude plus faible signifie que moins de bits sont nécessaires pour coder chaque échantillon du résidu. Comme le codage du résidu nécessite moins de bits que le codage du signal original et que le nombre de bits nécessaire au codage des coefficients n'est pas trés important, la taille du fichier compressé est alors inférieure à la taille du fichier original.

Dans notre travail, nous nous intéresserons à la deuxième étape : nous étudierons une autre méthode de décorrélation, la prédiction non linéaire, et nous verrons que dans certains cas, un codeur utilisant ce type de prédiction donne de meilleurs ratios de compression qu'un codeur utilisant une prédiction linéaire.

1.3 Le découpage en trames

La première étape consiste à découper le signal en trames. En effet, dans la mesure où tous les algorithmes calculent des paramètres dépendant des statistiques du signal, il est nécessaire que celui ci soit stationnaire, ou quasi stationnaire.

La taille de la trame sera constante pour éviter de rajouter sa longueur à l'en-tête de chaque trame ce qui peut être lourd en terme de débit. Les trames ne doivent être ni trop courtes (sinon l'en-tête de chaque trame risque d'être trop long par rapport à l'information audio), ni trop longues (car l'algorithme ne serait alors pas assez adaptatif). Typiquement, les codeurs existants travaillent sur des trames de 13 à 26 ms [2].

1.4 La décorrélation intra-canal

La deuxième étape consiste à enlever la redondance contenue dans une trame du canal audio. Pour réaliser cette étape, les codeurs existant mettent en oeuvre principalement deux techniques.

1.4.1 Par prédiction linéaire

Cette technique est la plus utilisée, elle consiste à prédire la valeur d'un échantillon $x(n)$ à partir d'une combinaison linéaire de K échantillons passés $x(n-k)$.

$$\widehat{x}(n) = -\sum_{k=1}^{K} a_k x(n-k) \tag{1.1}$$

$$e(n) = x(n) - \widehat{x}(n) \tag{1.2}$$

où $\widehat{x}(n)$ est la valeur prédite, $e(n)$ est l'erreur de prédiction et a_k sont les coefficients du prédicteur.

Idéalement, le prédicteur décorrèle le signal, par conséquent l'erreur de prédiction $e(n)$ est décorrélée et sa densité spectrale de puissance est constante (spectre blanc). Ces échantillons ayant un niveau généralement plus faible, ils nécessitent moins de bits pour être représentés. Le signal compressé est alors représenté par les paramètres du prédicteur et les échantillons du signal d'erreur.

Le prédicteur optimal est celui qui minimise la puissance de l'erreur de prédiction $E\left\{e(n)^2\right\}$. On cherche donc les coefficients a_k qui minimisent $E\left\{e(n)^2\right\}$, et ceci conduit au système d'équations linéaires de Yule-Walker :

$$\begin{pmatrix} r_x(0) & r_x(1) & \cdots & r_x(K-2) & r_x(K-1) \\ r_x(1) & \cdots & \cdots & \cdots & r_x(K-2) \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ r_x(K-2) & \cdots & \cdots & \cdots & r_x(1) \\ r_x(K-1) & r_x(K-2) & \cdots & r_x(1) & r_x(0) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{K-1} \\ a_K \end{pmatrix} = - \begin{pmatrix} r_x(1) \\ r_x(2) \\ \vdots \\ r_x(K-1) \\ r_x(K) \end{pmatrix} \tag{1.3}$$

où $r_x(k) = E\left\{x(n)x(n-k)\right\}$ est la fonction d'autocorrélation de $x(n)$. On remarque que la matrice d'autocorrélation est une matrice de Toeplitz symétrique. La résolution de ce système est alors couramment réalisé par l'algorithme de Levinson-Durbin. C'est un algorithme itératif (sur l'ordre du prédicteur) qui résout le système en un nombre restreint d'opérations sans calculer l'inverse de la matrice d'autocorrélation.

Augmenter l'ordre de prédiction K permet de diminuer de facon intéressante la variance de l'erreur de prédiction jusqu'à un certain seuil ; au delà, le gain n'est plus significatif. On peut quantifier l'efficacité du prédicteur linéaire en calculant le gain de prédiction G_P, qui est l'inverse du rapport de l'énergie de l'erreur de prédiction σ_e^2 sur l'énergie du signal original σ_x^2. C'est une mesure de la redondance à court terme dans le signal original dont la limite $G_{P,\max}$ dépend d'une mesure qui est fonction de la densité spectrale de puissance de x.

$$G_P = \frac{\sigma_x^2}{\sigma_e^2} \tag{1.4}$$

1.4.2 Par transformée orthogonale

Le codeur audio sans perte LTAC (Lossless Transform Audio Compression [3]) est le seul codeur existant utilisant une transformée orthogonale. Il se base sur un schéma similaire à celui d'algorithmes de compression avec perte :

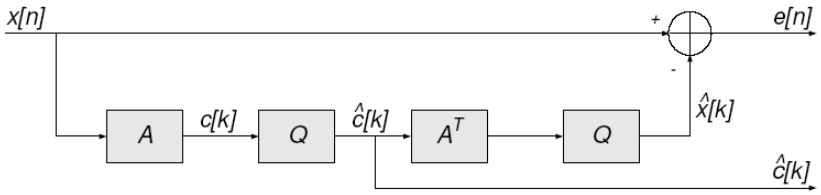


Fig. 1.2 - Structure d'un codeur par transformée

La transformation utilisée est une transformée en cosinus discrète (DCT), c'est une transformation orthogonale très utilisée et ses propriétés de répartition de l'énergie sont connues. Elle est représentée par la matrice A et son inverse est A^T. Le bloc Q est un bloc de quantification. Elle permet de reconstruire une bonne partie de l'information du signal à partir de peu de valeurs quantifiées. Les coefficients $c[k]$ de la transformée sont quantifiés en $\widehat{c}[k]$. Le signal compressé est donc constitué des $\widehat{c}[k]$ et du résidu $e[n]$, encodés par deux codes entropiques. Au décodage, les $\widehat{c}[k]$ permettent de calculer $\widehat{x}[n]$ et le résidu permet de retrouver $x[n] = \widehat{x}[n] + e[n]$.

1.5 La décorrélation inter-canaux

De rapides calculs d'intercorrélation entre les voies gauche et droite effectués sur des tranches de signaux musicaux hétérogènes permettent de se rendre compte de la forte corrélation qui peut exister entre les deux voies. La figure 1.3 permet de la visualiser pour 3 signaux de musique. Il s'agit de l'intercorrélation entre les 2 canaux d'une tranche de signal longue de 40 ms (soit typiquement 2 trames de calcul à 48 kHz). On y remarque immédiatement que l'intercorrélation est forte dans certains cas et qu'il serait dommage de ne pas exploiter cette redondance.

Des techniques permettant d'en tirer parti ont été développées dans le cadre des algorithmes de compression avec perte, comme par exemple l'intensity stéréo. Il existe aussi des méthodes paramètriques qui ne font que restituer une impression de stéréophonie. Plus d'informations sur ces techniques sont accessibles dans [4] et [5].

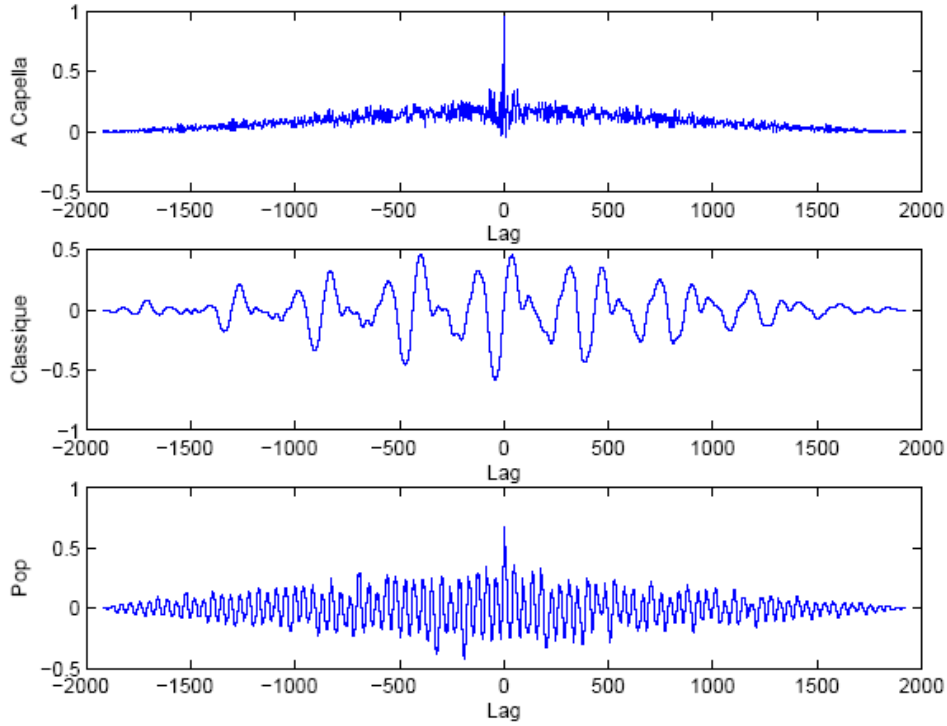


Fig. 1.3 - Intercorrélation des voies gauche et droite de 3 signaux audio

Les codeurs audio sans perte peuvent mettre en oeuvre une décorrélation inter-canaux rudimentaire en travaillant sur les signaux dits " mid " (c'est-à-dire l'information moyenne) et " side " (soit l'information latérale) :

$$m(t) = \frac{1}{2}\left(g(t) + d(t)\right) \tag{1.5}$$

$$s(t) = \frac{1}{2}\left(g(t) - d(t)\right) \tag{1.6}$$

où $g(t)$ et $d(t)$ sont respectivement les voies gauche et droite du signal audio stéréo. Ainsi, si les deux voies sont fortement corrélées à un même instant, le signal $s(t)$ sera très faible et le codage entropique très efficace. Le problème réside dans le fait que ce type d'opération ne tient pas compte d'un éventuel déphasage entre les 2 voies. Ainsi, la voie gauche et la voie droite peuvent être strictement identiques avec un certain déphasage, mais il n'y aura aucun gain lié à cette opération puisque le signal différence $s(t)$ ne sera pas nul. Cette technique ne sera donc pas adaptée au codage audio sans perte et d'autre solutions ont été envisagées, comme la prédiction linéaire stéréo.

1.6 La prédiction linéaire stéréo

Une technique de prédiction linéaire pouvant tirer parti des signaux stéréo a été adaptée au codage audio sans perte dans [6] et implémentée en 2002, par le groupe de recherche qui a développé les LTAC et LPAC : cette technique utilise la prédiction linéaire stéréo et a été étendue plus largement aux signaux multi-canaux [7]. L'intérêt de la prédiction linéaire stéréo est qu'elle effectue simultanément une décorrélation intra- et inter-canal.

Si on note $x_1(n)$ l'échantillon à l'instant n du signal de la voie gauche et $x_2(n)$ celui du signal de la voie droite, le prédicteur linéaire stéréo estime $x_1(n)$ à l'aide d'une combinaison linéaire des K_a valeurs passées $x_1(n-k)$ et d'une combinaison linéaire des K_b valeurs passées $x_2(n-k)$.

$$\widehat{x}_1(n) = \sum_{k=1}^{K_a} a_k x_1(n-k) + \sum_{k=1}^{K_b} b_k x_2(n-k) \tag{1.7}$$

L'ensemble des coefficients a_k constitue l'autoprédicteur de la voie x_1 et l'ensemble des coefficients b_k constitue l'interprédicteur de la voie x_1.

Comme dans le cas de la prédiction linéaire monophonique, on cherche à minimiser la variance de l'erreur de prédiction $e_1(n) = x_1(n) - \widehat{x}_1(n)$. Cette minimisation conduit à un système d'équations linéaire dont la matrice R_1 est formé de 4 sous-matrices de Toeplitz symétriques. Dans le cas particulier où on a $K_a = K_b$, le système peut être résolu avec l'algorithme de Levinson-Durbin multi-canaux [8]. Mais dans le cas $K_a \neq K_b$, cette algorithme ne peut pas être utilisé et on est obligé d'inverser la matrice. Sa forme symétrique permet d'utiliser la décomposition de Cholesky [7] pour réaliser l'inversion.

Pour la prédiction du signal de la voie droite, on peut tirer parti de l'entrelacement des voies x_1 et x_2 [6]. En effet, l'échantillon $x_1(n)$ qui vient d'être calculé par le décodeur peut-être utilisé pour prédire $x_2(n)$.

$$\widehat{x}_2(n) = \sum_{k=0}^{K_c} c_k x_1(n-k) + \sum_{k=1}^{K_d} d_k x_2(n-k) \tag{1.8}$$

La minimisation de l'erreur de prédiction $e_2(n)$ conduit aux même types d'équations et la matrice du système R_2 pourra aussi être inversée à l'aide d'une décomposition de Cholesky.

La figure 1.4 montre l'évolution du gain de prédiction en fonction de l'ordre pour un fichier de musique (chant à capella, les canaux gauche et droite sont très similaires). Dans le cas de l'analyse stéréo, l'ordre de l'autoprédicteur est fixé à 5 et l'ordre de l'interprédicteur incrémenté de 0 à 9.

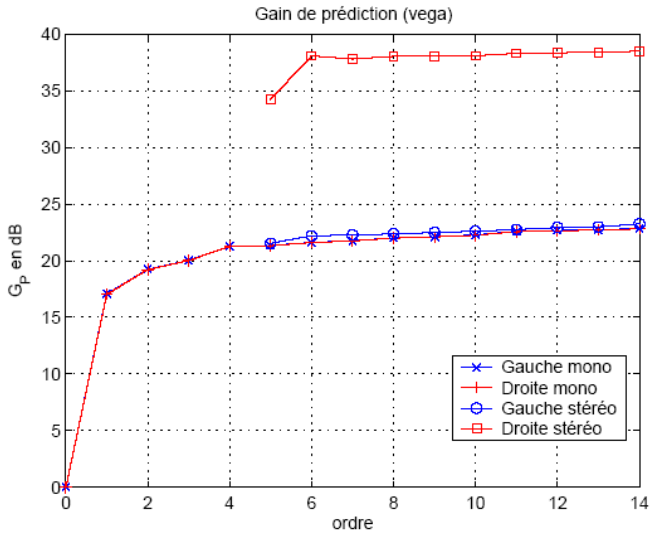


Fig. 1.4 - Gains de prédiction avec une prédiction LPC stéréo

En première constatation, on retrouve la similarité entre les 2 canaux, puisque les 2 prédicteurs linéaires mono donnent le même gain de prédiction. On peut alors noter l'apport de la prédiction stéréo par rapport à une simple analyse LPC :

- Une amélioration très faible sur le canal de gauche. En effet, les 2 voies étant quasi-similaires, les échantillons passés de la voie de droite n'apportent pas plus d'information que les échantillons passés de la voie de gauche.
- Un fort gain de 15 dB sur le canal de droite dés le premier niveau d'interprédiction. Ce résultat n'est pas surprenant dans la mesure où les voies gauche et droite étant très proche, la prédiction linéaire stéréo de $x_2[n]$ (canal de droite) sera très efficace puisqu'elle tire parti de l'échantillon $x_1[n]$.

1.7 Codage entropique

Le codage entropique est nécessaire afin de réduire le débit du résidu $e[n]$ et des coefficients de la transformée orthogonale dans le cas du LTAC. Le codage entropique le plus connu est le codage de Huffman mais il en existe beaucoup d'autres. On peut trouver une présentation du codage entropique en général dans [9].

1.7.1 Densité de probabilité du résidu

Le codage entropique du résidu de la prédiction dépend de la densité de probabilité de ses échantillons. La figure 1.5 extraite de [10] montre la répartition des résidus de prédiction. Les cercles ∘ représentent les valeurs observées. Ces courbes montrent que la distribution théorique se rapprochant le plus de la distribution des résidus de prédiction est la densité de probabilité laplacienne. Il existe

une famille de codeurs entropiques adapté aux alphabets distribués selon une loi laplacienne : il s'agit des codes de Golomb.

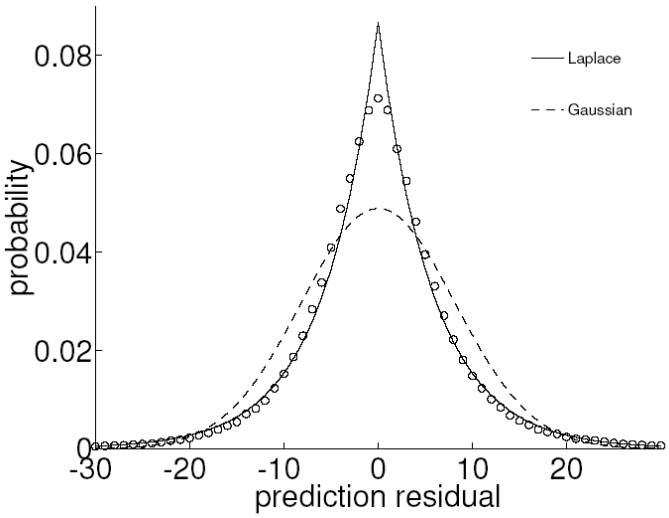


Fig. 1.5 - Densité de probabilité du résidu de prédiction linéaire

1.7.2 Le codage de Rice

Le codage de Rice est une forme particulière du codage de Golomb. Comme nous l'avons vu, c'est un code adapté lorsque les échantillons de faible amplitude sont très nombreux, ce qui est le cas du résidu de prédiction linéaire. Ce code, qui dépend d'un paramètre k, est un code de Golomb de paramètre $m = 2^k$. m est appelé paramètre de Golomb.

Les échantillons de faible amplitude sont intéressants car ils nécessitent peu de bits pour être codés. Prenons par exemple la série suivante :

$$\text{en base 10} \quad : \quad 10, 14, 15, 46 \tag{1.9}$$

$$\text{en base 2} \quad : \quad 1010, 1110, 1111, 101110 \tag{1.10}$$

Dans le cas d'un signal codé sur 16 bits, il faudrait concaténer 4 chaînes de 16 bits (soient 64 bits) pour représenter cette suite de 4 échantillons. L'idéal serait de concaténer les 4 chaînes en utilisant le nombre de bits nécessaires, 101011101111101110 (soient 18 bits). Mais une telle représentation bute sur un problème de taille : il est strictement impossible de dissocier les 4 échantillons. . . Le codage de Rice intervient alors à ce niveau en introduisant une technique pour concaténer des chaînes de longueurs différentes tout en permettant de bien les dissocier.

Le code de Rice d'un entier n est déterminé en trois parties, comme indiqué ci-dessous :

1. Un bit de signe.

2. Le code unitaire de $q = \left\lfloor \frac{n}{2^k} \right\rfloor$ (q 0 suivis d'un 1). $\lfloor\ \rfloor$ représente le quotient de la division eucildienne.
3. Les k bits de poids faible de n (reste de la division euclidienne).

Durant le processus, le paramètre k optimal peut être actualisé de différentes manières. On peut l'indexer sur la valeur moyenne d'un certain nombre d'échantillons passés, ou le calculer à l'aide de l'estimateur de m , donné dans [2] et [10] :

$$m = \log_2 \left(\ln(2) E \left\{ |e\left[n\right]| \right\} \right) \tag{1.11}$$

Reprenons la chaîne d'échantillons introduite plus haut afin d'illustrer le codage de Rice. On prendra $k = 4$ puisque 3 échantillons sur 4 peuvent être codés sur 4 bits.

1. Codage de $(10)_{10} = (1010)_2$
   - $sign(10) = 0$
   - $10/2^4 = 0$
   - 4 bits de poids faible : 1010

   Le code de $(10)_{10}$ est donc : 0|1|1010
2. Codage de $(14)_{10} = (1110)_2$
   - $sign(14) = 0$
   - $14/2^4 = 0$
   - 4 bits de poids faible : 1110

   Le code de $(14)_{10}$ est donc : 0|1|1110
3. Codage de $(15)_{10} = (1111)_2$
   - $sign(15) = 0$
   - $15/2^4 = 0$
   - 4 bits de poids faible : 1111

   Le code de $(15)_{10}$ est donc : 0|1|1111
4. Codage de $(46)_{10} = (101110)_2$
   - $sign(46) = 0$
   - $46/2^4 = 2$
   - 4 bits de poids faible : 1110

   Le code de $(46)_{10}$ est donc : 0|001|1110

La concaténation des 4 chaînes représente alors 26 bits contre 64 bits si on avait codé simplement les valeurs sur 16 bits, soit un ratio de compression supérieur à 2. Pour de plus amples précisions sur les codes de Rice Golomb, ici présentés simplement sous leur aspect technique, le lecteur pourra consulter [12] et [11].

Chapitre 2

Prédiction non linéaire avec un filtre de Volterra

On a vu dans la partie précédente que la majorité des codeurs audio sans pertes existants sont basés sur une prédiction linéaire. La prédiction linéaire permet d'obtenir un résidu dont l'énergie est plus faible que le signal original. Ce résidu peut alors être représenté par un nombre de bits moins important grâce à un codage entropique adapté à la distribution laplacienne du résidu : le codage de Rice. On a vu aussi que le gain de prédiction d'une prédiction linéaire augmentait avec l'ordre jusqu'à un certain seuil. Pour dépasser ce seuil, il faudra donc envisager d'autre types de prédiction : des prédictions non linéaires.

On étudiera dans un premier temps une prédiction non linéaire basée sur un filtre de Volterra. Ce type de prédiction a été étudié dans le cas d'un codeur de parole avec pertes [13] [15]. Les conclusions de ces articles étaient les suivantes : la prédiction non linéaire donne de meilleurs gains de prédiction qu'une simple prédiction linéaire mais il existe des problèmes de stabilité au niveau du décodeur, ceci étant du à la quantification du résidu (même si le prédicteur n'est pas quantifié). Or pour un codeur sans perte, le résidu n'est pas quantifié, ce problème ne se posera donc pas dans notre cas.

2.1 Le filtre de Volterra

Soit $x(n)$ le signal d'entrée du filtre de Volterra et $y(n)$ le signal de sortie. La relation qui lie $x(n)$ et $y(n)$ est :

$$y(n) = \sum_{k=1}^{N} \left[\sum_{i_1=1}^{P} ... \sum_{i_k=1}^{P} h_k(i_1, ..., i_k).x(n-i_1)...x(n-i_k) \right] \tag{2.1}$$

où $h_k(m_1, ..., m_k)$ sont les coefficients du filtre. N représente l'ordre de la non linéarité, P représente l'ordre du filtre.

Dans notre cas, on utilise le filtre de Volterra comme un prédicteur : on cherche à prédire $x(n)$ à partir de ses valeurs précédentes $x(n-i)$, $i = 1, .., P$ et $k = 1, .., N$. Pour chaque trame, on va donc chercher les coefficients du filtre qui minimisent l'erreur quadratique moyenne $E\left[(x(n) - y(n))^2\right]$. On observera que la sortie du filtre est linéaire vis-à-vis de chaque coefficient du filtre de Volterra. En vertu de cette propriété, l'erreur quadratique moyenne ne possède qu'un seul minimum global et nous pouvons le calculer analytiquement. Mais la complexité augmente fortement avec l'ordre de la non-linéarité. On se limitera donc à l'ordre 2.

$$y(n) = \sum_{i=1}^{P} h_1(i)x(n-i) + \sum_{i=1}^{P}\sum_{j=1}^{P} h_2(i,j)x(n-i)x(n-j) \tag{2.2}$$

2.1.1 Résolution dans le cas général

On cherche à minimiser l'erreur quadratique $E\left[(x(n) - y(n))^2\right]$. Ce problème a été résolu dans [13]. Ecrivons (2.2) sous une forme vectorielle :

$$\begin{aligned} y(n) &= \sum_{i=1}^{P} h_1(i)x(n-i) + \sum_{i=1}^{P}\sum_{j=1}^{P} h_2(i,j)x(n-i)x(n-j) \\ &= \mathbf{h}^T\mathbf{x}_n \end{aligned} \tag{2.3}$$

$$\text{avec} \quad \mathbf{h} = \begin{pmatrix} h_1(1) \\ \vdots \\ h_1(P) \\ h_2(1,1) \\ h_2(1,2) \\ \vdots \\ h_2(1,P) \\ h_2(2,1) \\ \vdots \\ h_2(2,P) \\ \vdots \\ h_2(P,P) \end{pmatrix} \quad ; \quad \mathbf{x}_n = \begin{pmatrix} x(n-1) \\ \vdots \\ x(n-P) \\ x(n-1)x(n-1) \\ x(n-1)x(n-2) \\ \vdots \\ x(n-1)x(n-P) \\ x(n-2)x(n-1) \\ \vdots \\ x(n-2)x(n-P) \\ \vdots \\ x(n-P)x(n-P) \end{pmatrix} \tag{2.4}$$

L'erreur de prédiction est donnée par :

$$\begin{aligned} e^2 &= E\left(e(n)^2\right) && (2.5)\\ &= E\left((x(n)-y(n))^2\right) && (2.6)\\ &= E(x(n)^2) - 2E(x(n)\mathbf{h}^T\mathbf{x}_n) + E(\mathbf{h}^T\mathbf{x}_n\mathbf{h}^T\mathbf{x}_n) && (2.7)\\ &= E(x(n)^2) - 2\mathbf{h}^T E(x(n)\mathbf{x}_n) + \mathbf{h}^T E(\mathbf{x}_n^T\mathbf{x}_n)\mathbf{h} && (2.8)\\ &= E(x(n)^2) - 2\mathbf{h}^T\mathbf{m} + \mathbf{h}^T\mathbf{M}\mathbf{h} && (2.9) \end{aligned}$$

avec

$$\mathbf{m} = E(x(n)\mathbf{x}_n) \quad \text{et} \quad \mathbf{M} = E(\mathbf{x}_n^T\mathbf{x}_n) \tag{2.10}$$

Les coefficients optimaux $\mathbf{h}_{opt}$ qui minimisent l'erreur quadratique vérifient :

$$\left(\frac{\partial(e^2)}{\partial\mathbf{h}}\right)_{\mathbf{h}=\mathbf{h}_{opt}} = 0 \tag{2.11}$$

$$\Leftrightarrow -2\mathbf{m} + 2\mathbf{M}\mathbf{h}_{opt} = 0 \tag{2.12}$$

ce qui donne la solution optimale pour les coefficients du filtre

$$\mathbf{h}_{opt} = \mathbf{M}^{-1}\mathbf{m} \tag{2.13}$$

2.1.2 Simplifications

Etant donné que l'on a

$$h_2(i,j) = h_2(j,i) \tag{2.14}$$

On peut réécrire (2.2) sous la forme :

$$\begin{aligned} y(n) &= \sum_{i=1}^{P} h_1(i)x(n-i) + \sum_{i=1}^{P}\sum_{j=i}^{P} h_2(i,j).x(n-i)x(n-j)\\ &= \mathbf{h}^T\mathbf{x}_n && (2.15) \end{aligned}$$

avec :

$$\mathbf{h} = \begin{pmatrix} h_1(1) \\ \vdots \\ h_1(P) \\ h_2(1,1) \\ h_2(1,2) \\ \vdots \\ h_2(1,P) \\ h_2(2,2) \\ \vdots \\ h_2(2,P) \\ \vdots \\ h_2(P,P) \end{pmatrix} \quad ; \quad \mathbf{x}_n = \begin{pmatrix} x(n-1) \\ \vdots \\ x(n-P) \\ x(n-1)x(n-1) \\ x(n-1)x(n-2) \\ \vdots \\ x(n-1)x(n-P) \\ x(n-2)x(n-2) \\ \vdots \\ x(n-2)x(n-P) \\ \vdots \\ x(n-P)x(n-P) \end{pmatrix}$$

On a alors $\frac{P(P+3)}{2}$ coefficients au lieu de $P(P+1)$.

2.2 Prédiction non linéaire stéréo avec un filtre de Volterra

Afin d'exploiter la corrélation entre les 2 canaux du signal stéréo, on a vu dans 1.5 qu'une prédiction linéaire stéréo donnait de bons résultats. Cette technique peut être étendue aux ordres supérieurs grâce à un filtre de Volterra ayant pour entrées les valeurs passées de $x_1(n)$ et $x_2(n)$.

2.2.1 Prédiction du canal de gauche

Pour prédire $x_1(n)$, on va utiliser un filtre de Volterra qui a pour entrées les valeurs passées de $x_1(n)$ et $x_2(n)$:

$$\begin{aligned} \widehat{x}_1(n) &= \sum_{i=1}^{P_1} a_1(i) x_1(n-i) + \sum_{i=1}^{P_{1NL}} \sum_{j=i}^{P_{1NL}} a_2(i,j) x_1(n-i)\, x_1(n-j) \\ &\quad + \sum_{i=1}^{P_2} b_1(i) x_2(n-i) + \sum_{i=1}^{P_{2NL}} \sum_{j=i}^{P_{2NL}} b_2(i,j) x_2(n-i)\, x_2(n-j) \end{aligned} \tag{2.16}$$

que l'on peut écrire sous forme vectorielle

$$\widehat{x}_1(n) = \mathbf{h}_1^T \mathbf{x}_{1n} \tag{2.17}$$

avec

$$\mathbf{h}_1 = \begin{pmatrix} a_1(1) \\ \vdots \\ a_1(P_1) \\ a_2(1,1) \\ \vdots \\ a_2(1,P_{1NL}) \\ a_2(2,2) \\ \vdots \\ a_2(2,P_{1NL}) \\ \vdots \\ a_2(P_{1NL},P_{1NL}) \\ b_1(1) \\ \vdots \\ b_1(P_2) \\ b_2(1,1) \\ \vdots \\ b_2(P_{2NL},P_{2NL}) \end{pmatrix} ; \mathbf{x}_{1n} = \begin{pmatrix} x_1(n-1) \\ \vdots \\ x_1(n-P_1) \\ x_1(n-1)x_1(n-1) \\ \vdots \\ x_1(n-1)x_1(n-P_{1NL}) \\ x_1(n-2)x_1(n-2) \\ \vdots \\ x_1(n-2)x_1(n-P_{1NL}) \\ \vdots \\ x_1(n-P_{1NL})x_1(n-P_{1NL}) \\ x_2(n-1) \\ \vdots \\ x_2(n-P_2) \\ x_2(n-1)x_2(n-1) \\ \vdots \\ x_2(n-P_{2NL})x_2(n-P_{2NL}) \end{pmatrix}$$

La solution optimale pour les coefficients du filtre est alors

$$\mathbf{h}_{1opt} = \mathbf{M}_1^{-1}\mathbf{m}_1 \tag{2.18}$$

avec

$$\mathbf{m}_1 = E(x_1[n]\,\mathbf{x}_{1n}) \quad \text{et} \quad \mathbf{M}_1 = E(\mathbf{x}_{1n}^T\mathbf{x}_{1n}) \tag{2.19}$$

2.2.2 Prédiction du canal de droite

Pour la prédiction du canal de droite, on peut tirer parti de l'entrelacement des voies x_1 et x_2. En effet, l'échantillon $x_1[n]$ qui vient d'être calculé par le décodeur peut-être utilisé pour prédire $x_2[n]$.

$$\begin{aligned} \widehat{x}_2[n] &= \sum_{i=1}^{P_1} c_1(i)x_2[n-i] + \sum_{i=1}^{P_{1NL}}\sum_{j=i}^{P_{1NL}} c_2(i,j)x_2[n-i]\,x_2[n-j] \\ &\quad + \sum_{i=0}^{P_2} d_1(i)x_1[n-i] + \sum_{i=0}^{P_{2NL}}\sum_{j=i}^{P_{2NL}} d_2(i,j)x_1[n-i]\,x_1[n-j] \end{aligned} \tag{2.20}$$

qui peut s'écrire sous la forme vectorielle

$$\widehat{x}_2\left[n\right] = \mathbf{h}_2^T \mathbf{x}_{2n} \tag{2.21}$$

La solution optimale pour les coefficients du filtre est alors

$$\mathbf{h}_{2opt} = \mathbf{M}_2^{-1} \mathbf{m}_2 \tag{2.22}$$

avec

$$\mathbf{m}_2 = E(x_2\left[n\right] \mathbf{x}_{2n}) \quad \text{et} \quad \mathbf{M}_2 = E(\mathbf{x}_{2n}^T \mathbf{x}_{2n}) \tag{2.23}$$

Nombre de coefficients du filtre et choix des ordres :

Le nombre de coefficients pour le canal de gauche est

$$N_1 = P_1 + \frac{\left(P_{1NL}+1\right) P_{1NL}}{2} + P_2 + \frac{\left(P_{2NL}+1\right) P_{2NL}}{2} \tag{2.24}$$

et le nombre de coefficients du canal de droite est

$$N_2 = N_1 + P_{2NL} + 2 \tag{2.25}$$

Dans [1], il a été retenu les valeurs $P1 = 20$ et $P2 = 10$ pour la prédiction linéaire stéréo. Nous choisirons donc dans l'étude qui suit : $P1 = 20$, $P2 = 10$ et $P_{1NL} = P_{2NL} = 10$, afin de comparer avec les résultats obtenus pour la prédiction linéaire stéréo. Les valeurs de P_{1NL} et P_{2NL} ne sont pas choisies trop grandes pour ne pas avoir un nombre trop important de coefficients.

Pour $P_1 = 20$, $P_2 = 10$ et $P_{1NL} = P_{2NL} = 10$, on a :

$$N_1 = 140 \text{ et } N_2 = 152$$

(Alors que pour $P_1 = P_{1NL} = 20$ et $P_2 = P_{2NL} = 10$, on aurait : $N_1 = 295$ et $N_2 = 307$)

2.2.3 Approche forward

On étudie dans un premier temps une approche forward du prédicteur : la trame utilisée pour déterminer les paramètres du prédicteur est celle sur laquelle on applique le prédicteur. C'est l'approche naturellement utilisé en codage audio sans perte. Le fichier compressé est alors composé :

- du résidu
- des paramètres de la prédiction

Estimation de $\mathbf{m}_1, \mathbf{m}_2, \mathbf{M}_1$ et $\mathbf{M}_2$:

Les vecteurs et matrices $\mathbf{m}_1, \mathbf{m}_2, \mathbf{M}_1$ et $\mathbf{M}_2$ sont estimés sur une trame étendue afin d'augmenter le nombre d'échantillons utilisés et ainsi améliorer la précision de l'estimation. Ces trames étendues sont pondérées par une fenêtre de Hamming (voir fig 2.1).

$$\mathbf{m}_1 = \frac{1}{N-n_0} \sum_{n=1+n_0}^{N} x_{1_{et.}}[n]\, \mathbf{x}_{1_{et.}n} \tag{2.26}$$

$$\mathbf{M}_1 = \frac{1}{N-n_0} \sum_{n=1+n_0}^{N} \mathbf{x}_{1_{et.}n}^T \mathbf{x}_{1_{et.}n} \tag{2.27}$$

$$\mathbf{m}_2 = \frac{1}{N-n_0} \sum_{n=1+n_0}^{N} x_{2_{et.}}[n]\, \mathbf{x}_{2_{et.}n} \tag{2.28}$$

$$\mathbf{M}_2 = \frac{1}{N-n_0} \sum_{n=1+n_0}^{N} \mathbf{x}_{2_{et.}n}^T \mathbf{x}_{2_{et.}n} \tag{2.29}$$

avec N le nombre d'échantillon de la trame étendue

et $n_0 = \max(P_1, P_2, P_{1NL}, P_{2NL})$.

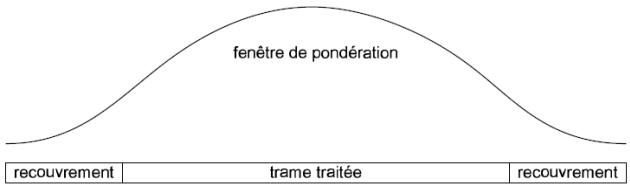


Fig. 2.1 - Trame étendue

Inversion des matrices $\mathbf{M}_1$ et $\mathbf{M}_2$

L'inversion des matrices $\mathbf{M}_1$ et $\mathbf{M}_2$ est réalisée à l'aide d'une décomposition de Cholesky. Or, pour utiliser cette technique, il faut que les matrices soient définies positives. Comme il se peut que certaines valeurs propres soient nulles, il est nécessaire de conditionner les matrices. Pour cela, on ajoutera un δ sur la diagonale de la matrice. Dans le cas de la prédiction linéaire, ce δ correspond à l'énergie d'un bruit fictif dont le niveau est 40dB plus faible que la voie la plus puissante. On prendra la même valeur de δ pour les matrices $\mathbf{M}_1$ et $\mathbf{M}_2$.

Mise en oeuvre

L'algorithme a été testé sur 16 fichiers audio : 12 fichiers d'une base de donnée de test MPEG, deux fichiers de parole convertis à 8khz (engl8 et germ8) et deux fichiers de musique rock (clapton et u2_court). On a choisi les ordres suivants : $P_1 = 20$, $P_2 = 10$ et $P_{1NL} = P_{2NL} = 10$. On a pris une longueur de trame de 20ms (882 échantillons à 44,1 kHz) et un recouvrement de 5ms. On a alors calculé le gain de prédiction sur chaque trame des voies gauche et droite de chaque fichier.

La Figure 2.2 montre la différence avec le gain de prédiction obtenu avec un prédicteur linéaire stéréo (minimum, maximum et moyenne). Le prédicteur linéaire stéréo utilise le même algorithme mais sans la partie non linéaire ($P_{1NL} = P_{2NL} = 0$). La dernière colonne du tableau correspond au gain apporté par la technique qui consiste à choisir la prédiction qui donne le meilleur gain de prédiction.

		ΔGp min	*ΔGp max*	*ΔGp moyenne*	*ΔGp moyenne 2*
bagp	Gauche	-0.00	5.24	2.08	2.08
	Droite	0	3.13	1.17	1.17
cast	Gauche	-7.70	7.11	0.19	0.36
	Droite	-12.34	6.96	0.15	0.34
clapton	Gauche	-0.16	1.56	0.24	0.24
	Droite	-0.91	1.45	0.16	0.16
engl	Gauche	-0.39	0.72	0.10	0.10
	Droite	-6.63	0.60	-0.42	0.01
engl8	Gauche	-4.34	5.19	0.89	0.95
	Droite	-18.01	0.78	-3.17	0.03
germ	Gauche	-0.46	0.83	0.13	0.14
	Droite	-4.22	2.46	0.15	0.31
germ8	Gauche	-5.13	8.54	1.41	1.45
	Droite	-8.95	7.37	0.79	1.26
gloc	Gauche	-8.20	1.86	-0.19	0.16
	Droite	-5.36	2.96	0.01	0.22
harp	Gauche	-3.81	1.77	0.30	0.31
	Droite	-7.97	1.99	0.31	0.33
orch	Gauche	-0.39	0.32	0.02	0.04
	Droite	-0.27	0.56	0.02	0.04
pitc	Gauche	-0.06	6.03	1.58	1.58
	Droite	-0.03	4.63	1.73	1.73
popm	Gauche	-0.56	1.75	0.32	0.32
	Droite	-0.36	1.31	0.21	0.21
stri	Gauche	-0.24	1.17	0.04	0.05
	Droite	-1.02	1.25	0.06	0.06
trum	Gauche	-1.40	1.77	0.20	0.23
	Droite	-0.68	1.70	0.13	0.17
u2_court	Gauche	-0.21	1.60	0.17	0.17
	Droite	-0.43	1.62	0.16	0.17
vega	Gauche	-0.54	1.02	0.09	0.10
	Droite	-0.56	2.34	0.08	0.09

Fig. 2.2 - Différence entre les gains de prédiction obtenus avec une prédiction non linéaire et les gains de prédiction obtenus avec une prédiction linéaire ($\Delta > 0 \Longrightarrow G_{Pnonlin} > G_{Plin}$) pour une prédiction stéréo forward

On remarque tout d'abord que le gain par rapport au prédicteur linéaire dépend beaucoup du type de signal audio. Par exemple, le fichier Bagp, un signal audio de cornemuse, obtient un gain moyen sur toutes les trames de 2.1 dB pour la voie gauche et de 1.2 dB pour la voie droite. Alors que le fichier Orch, un enregistrement d'orchestre symphonique, obtient un gain de 0.02 dB pour les voies gauche et droite.

On remarque ensuite que pour des fichiers de parole, on obtient de bien meilleurs résultats en bande étroite (sauf pour le canal droit de engl). La figure 2.3 montre une partie voisée du canal gauche du signal de parole germ (échantillonné à 48 khz) ainsi que les résidus des prédictions stéréo linéaire et non linéaire. La figure 2.4 montre la même partie voisée du signal mais échantillonné à

8 khz (germ8) ainsi que les résidus des prédictions. Le gain apporté par la prédiction non linéaire est significatif pour le signal échantillonné à 8khz (gain entre 3 et 7 dB par rapport à la prédiction linéaire). Mais on peut remarquer ici que pour un signal de parole à 8khz, une prédiction à long terme serait certainement au moins aussi efficace que la prédiction non linéaire.

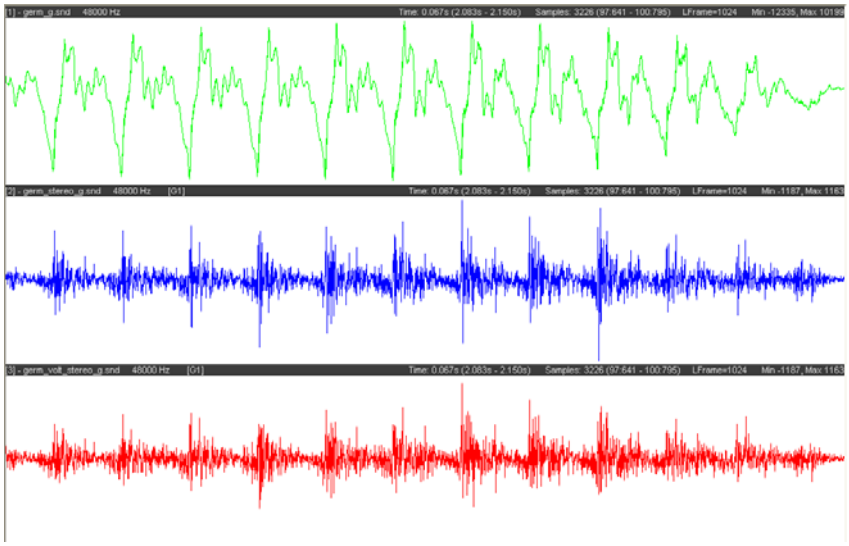


Fig. 2.3 - (de haut en bas) signal de parole échantilloné à 48 khz, résidu de la prédiction linéaire et résidu de la prédiction non linéaire (les résidus sont à la même échelle)

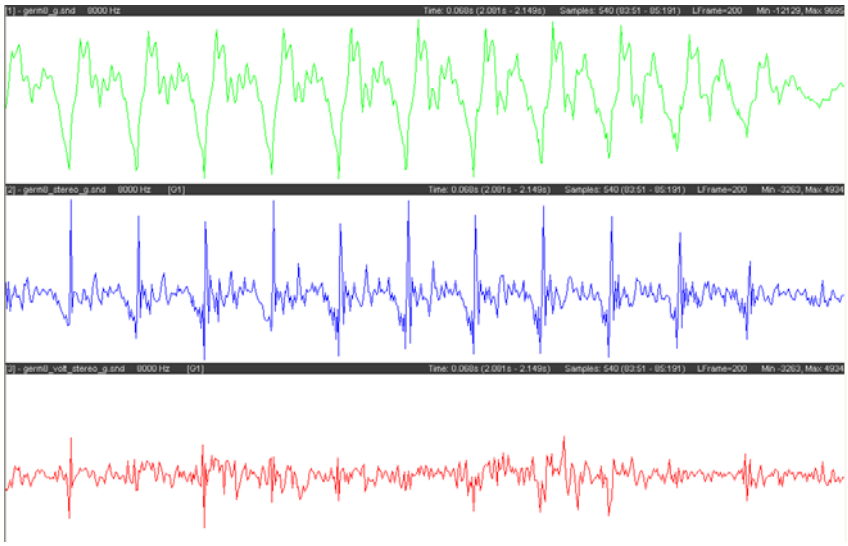


Fig. 2.4 - (de haut en bas) signal de parole échantilloné à 8 khz, résidu de la prédiction linéaire et résidu de la prédiction non linéaire (les résidus sont à la même échelle)

La figure suivante est un extrait du fichier cast (castagnette) ainsi que les résidus des prédictions linéaire et non linéaire. On voit bien que la prédiction non linéaire réduit l'amplitude des "claquements" de la castagnette (gain de 5 à 7 dB par rapport à la prédiction linéaire).

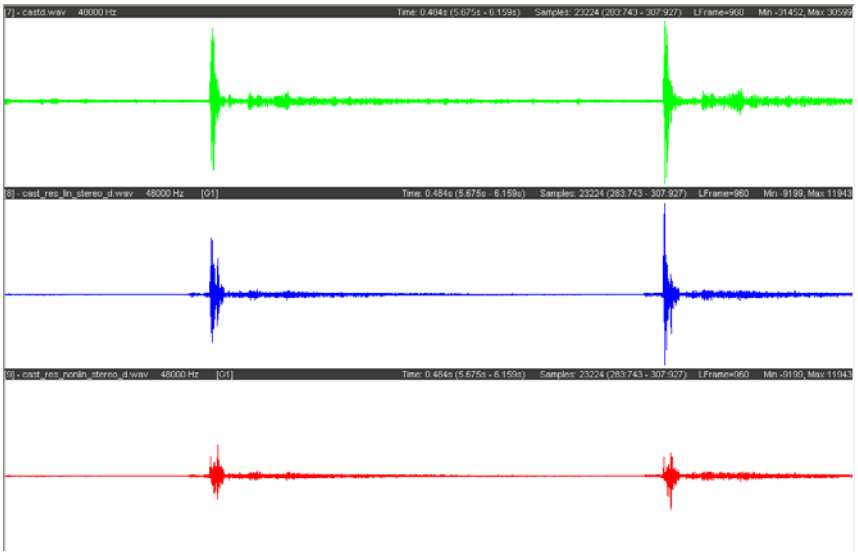


Fig. 2.5 - (de haut en bas) signal de castagnette échantilloné à 48 khz, résidu de la prédiction linéaire et résidu de la prédiction non linéaire (les résidus sont à la même échelle)

Les figures suivantes sont les histogrammes des gains du prédicteur non linéaire par rapport au prédicteur linéaire par trame pour deux fichiers : clapton et vega. On remarque que pour une petite proportion des trames, le prédicteur non linéaire se comporte moins bien que le prédicteur linéaire, c'est pourquoi la technique qui consiste à choisir, pour chaque trame, la prédiction qui donne le meilleur gain de prédiction peut améliorer les résultats.

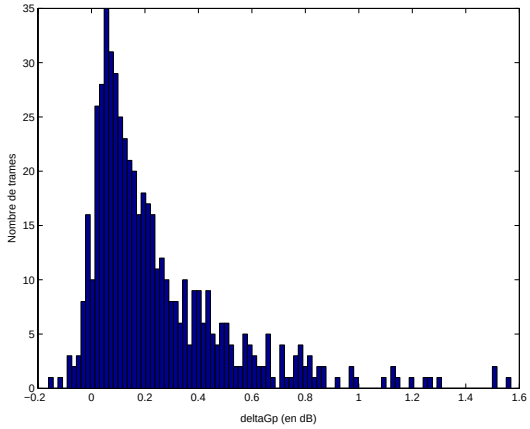


Fig. 2.6 - Histogramme de ΔG_p pour la voie gauche de clapton

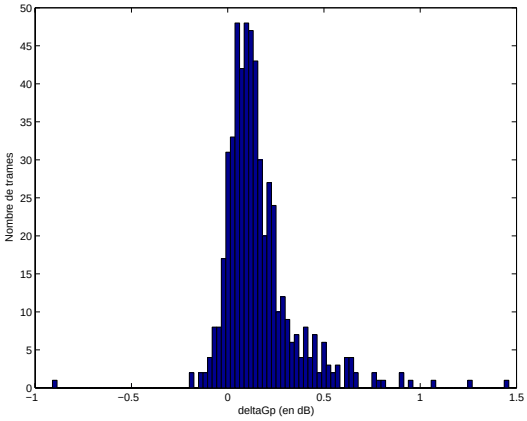


Fig. 2.7 - Histogramme de ΔG_p pour la voie droite de clapton

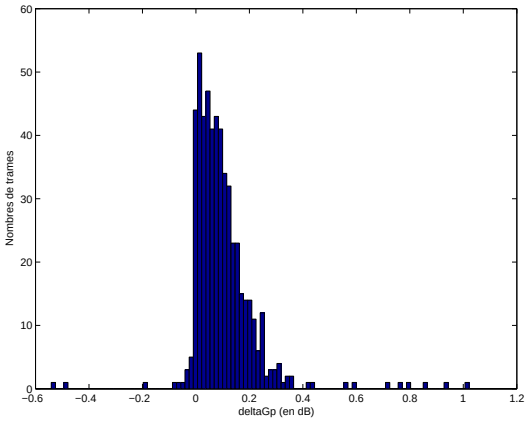


Fig. 2.8 - Histogramme de ΔG_p pour la voie gauche de vega

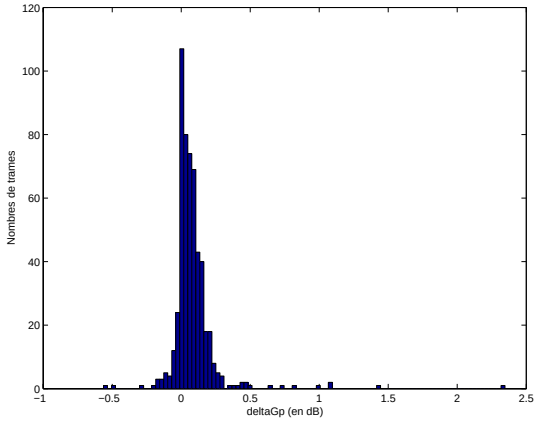


Fig. 2.9 - Histogramme de ΔG_p pour la voie droite de vega

Bilan

Si on fait un bilan sur tous les fichiers, on obtient en moyenne un gain de 0.48dB pour la voie gauche et 0.1dB sur la voie droite. Lorsqu'on choisit la meilleure prédiction, on obtient en moyenne un gain de 0.52dB pour la voie gauche et 0.39dB sur la voie droite. Le gain n'est donc pas très important en moyenne.

	canal gauche	canal droite
Gain par rapport au linéaire	0.48dB	0.1dB
Gain par rapport au linéaire (choix meilleur prédiction)	0.52dB	0.39dB

Tab. 2.1 - Moyenne des gains - prédiction stéréo forward

Quantification des coefficients

Dans l'approche forward, il faut considérer la quantification des coefficients. Or il a été montré dans [7] que les techniques actuelles de quantification des coefficients du prédicteur LPC ne pouvait pas s'appliquer à la prédiction linéaire stéréo. La solution avait été d'utiliser une quantification scalaire uniforme sur 12 bits. De plus, étant donné que la dynamique des coefficients peut varier énormément d'une trame à l'autre, une quantification adaptative avait été adoptée.

Le prédicteur linéaire n'étant qu'un cas particulier du prédicteur utilisant un filtre de Volterra (filtre d'ordre 1), il doit en être de même pour le prédicteur non linéaire. Pour chaque trame on va donc réaliser les opérations suivantes :

– Recherche du coefficient dont l'amplitude en valeur absolue est maximale $c_{\max}$

– Calcul du facteur d'échelle F qui est le rapport entre la plus grande valeur codable 2^{N_b} (où N_b est le nombre de bits de quantifications) et le plus grand coefficient $c_{\max}$.

$$F = \frac{2^{N_b}}{c_{\max}}$$

Remarquons que si le coefficient le plus grand ne dépasse pas l'unité en valeur absolue, le facteur d'échelle est automatiquement égal à la plus grande valeur codable.

– Tous les coefficients sont alors multipliés par ce facteur d'échelle et arrondis à l'entier le plus proche.

La figure suivante montre le gain de prédiction obtenu sur les 2 canaux du fichier clapton en fonction du nombre de bits de quantification. On remarque que pour obtenir le même gain de prédiction qu'une prédiction sans quantification à 10^{-2} prés, il est nécessaire de quantifier les coefficients sur 12 bits.

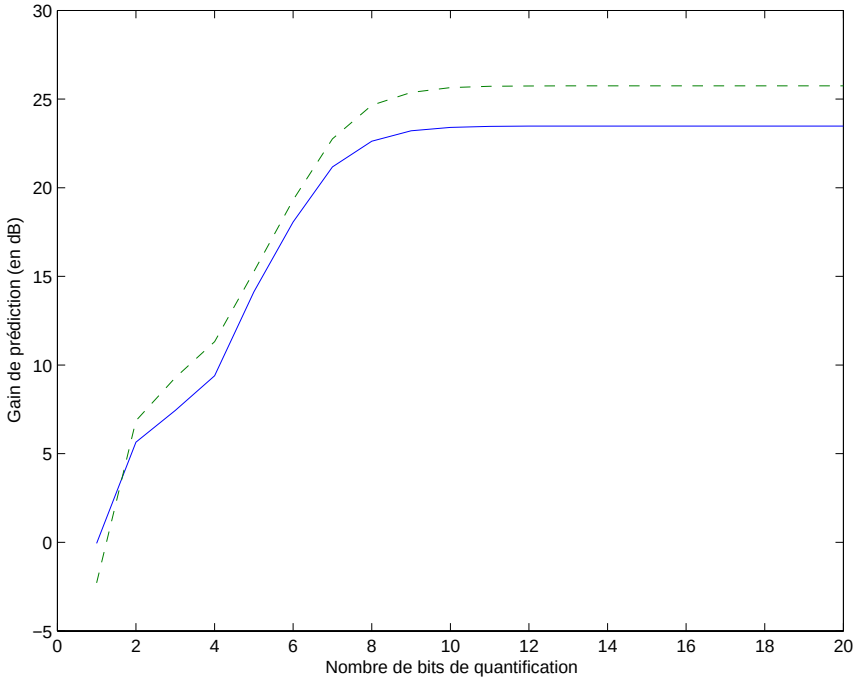


Fig. 2.10 - Gain de prédiction en fonction du nombre de bits de quantification pour la voie gauche (trait plein) et la voie droite (trait pointillé) du fichier clapton

De plus, le prédicteur non linéaire a un nombre très important de coefficients à transmettre : avec $P_1 = 20$, $P_2 = 10$ et $P_{1NL} = P_{2NL} = 10$, le prédicteur non linéaire a 231 coefficients de plus que le prédicteur linéaire (292 au lieu de 61). Pour des trames de 882 échantillons (20ms à 44,1khz) le débit lié aux coefficients serait alors de 1.98 bits/échantillons contre 0.41 bits/échantillons (si les coefficients sont codés sur 12 bits).

Le gain apporté par le non linéaire, qui n'est pas très important, sera donc perdu au final à cause du nombre important de coefficients à transmettre.

On envisagera donc dans ce qui suit une configuration backward, pour ne pas avoir à transmettre les coefficients.

2.2.4 Approche backward

L'approche backward a été introduite dans [16] afin de réduire le délai de codage. Dans ce type d'analyse, les coefficients du prédicteur ne sont pas calculés sur la trame de signal à coder, mais sur la trame précédente. Ainsi, le décodeur disposant des échantillons de la trame précédente, il peut calculer les coefficients du prédicteur en utilisant la mëme méthode que le codeur. On voit alors qu'il n'y a plus de coefficients de prédiction à transmettre. Enfin, il n'est pas nécessaire de stocker une trame de signal pour faire une nouvelle analyse. Cette technique a donné naissance au LD-CELP (Low-Delay CELP), un codeur de parole à 16 kb/s à faible délai d'encodage. La perte de trame constitue le principal défaut de ce codeur. On imagine les difficultés à faire une analyse LPC backward lorsque la trame est perdue ou erronée.

L'optique d'une prédiction linéaire stéréo portée en backward est par conséquent séduisante : pas de coefficients à transmettre,

- donc pas de quantiffication à réaliser
- pas de limite dans le choix des ordres de prédiction (si ce n'est le temps de calcul)
- enfin on peut réaliser autant d'analyse que nécessaire.

Le seul inconvénient étant une diminution du gain de prédiction par rapport à l'approche forward, puisque les coefficients seront calculés sur la trame passée.

Mise en oeuvre :

La longueur des trames est aussi de 20ms mais le recouvrement (5ms) n'est que d'un seul coté étant donné que l'analyse se fait sur les échantillons passés. On choisit dans cette approche de pondérer la trame étendue par une fenêtre asymétrique afin de privilégier les échantillons les plus récents : 95% de la fenêtre correspond à la moitié d'une fenêtre de Hamming et la partie restante est un quart de période d'une fonction cosinus.

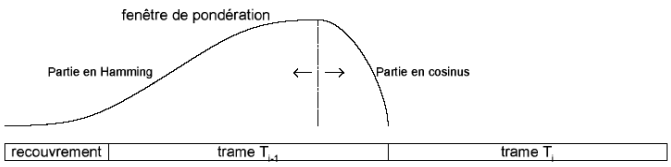


Fig. 2.11 - Trame étendue

Pour que le prédicteur soit plus adaptatif, on décide d 'effectuer une analyse en sous-trames : le prédicteur est mis à jour 4 fois par trames, le prédicteur restant identique sur des sous-trames de

5ms.

Le tableau suivant (figure 2.11)montre le gain en dB par rapport à un prédicteur linéaire stéréo. Le prédicteur linéaire stéréo utilise le même algorithme mais sans la partie non-linéaire.

		ΔGp min	*ΔGp max*	*ΔGp moyenne*	*ΔGp moyenne 2*
bagp	Gauche	-1.83	5.48	1.88	1.89
	Droite	-2.43	3.19	0.86	0.89
cast	Gauche	-12.56	6.08	-0.17	0.13
	Droite	-6.09	4.36	-0.16	0.10
clapton	Gauche	-11.11	1.47	-0.42	0.03
	Droite	-6.04	2.52	-0.53	0.01
engl	Gauche	-13.53	2.26	-0.13	0.07
	Droite	-28.57	3.48	-0.84	0.03
engl8	Gauche	-20.20	9.11	-0.28	0.39
	Droite	-22.75	13.04	-3.29	0.06
germ	Gauche	-15.61	3.02	-0.09	0.08
	Droite	-22.78	4.43	-0.15	0.32
germ8	Gauche	-21.73	7.81	0.04	0.63
	Droite	-25.73	9.76	-0.21	0.96
gloc	Gauche	-18.08	5.44	-0.64	0.23
	Droite	-11.96	7.91	-0.36	0.30
harp	Gauche	-4.35	1.80	-0.02	0.11
	Droite	-3.19	1.51	-0.00	0.12
orch	Gauche	-2.953	0.58	-0.31	0.00
	Droite	-3.09	0.60	-0.36	0.00
pitc	Gauche	-1.20	6.34	1.55	1.55
	Droite	-1.70	5.07	1.68	1.69
popm	Gauche	-6.29	1.57	-0.26	0.06
	Droite	-8.88	1.32	-0.38	0.02
stri	Gauche	-5.86	0.39	-0.29	0.00
	Droite	-6.03	1.69	-0.24	0.00
trum	Gauche	-4.66	3.30	-0.32	0.15
	Droite	-6.24	2.72	-0.46	0.09
u2_court	Gauche	-6.66	1.34	-0.51	0.01
	Droite	-6.28	0.86	-0.57	0.00
vega	Gauche	-14.93	0.77	-0.10	0.03
	Droite	-17.29	1.68	-0.18	0.02

Fig. 2.12 - Différence entre les gains de prédiction obtenus avec une prédiction non linéaire et les gains de prédiction obtenus avec une prédiction linéaire ($\Delta > 0 \Longrightarrow G_{Pnonlin} > G_{Plin}$) pour une prédiction stéréo backward

On remarque que le gain est toujours négatif en moyenne sur chaque trame sauf pour deux fichiers : bagp et pitc, et pour le canal gauche de germ8.

Même si deux trames consécutives sont très similaires, il existe de petites différences qui perturbent le prédicteur. En effet, prédire une trame avec un prédicteur dont l'analyse a été faite sur la trame précédente peut être vu comme une prédiction forward auquel on a fait subir une variation minime des coefficients. Or, le prédicteur non linéaire est beaucoup plus sensible à cette variation que le prédicteur linéaire, ce qui explique les résultats obtenus.

Les figures suivantes sont les histogrammes des gains du prédicteur non linéaire par rapport au prédicteur linéaire par trame pour deux fichiers : clapton et vega.

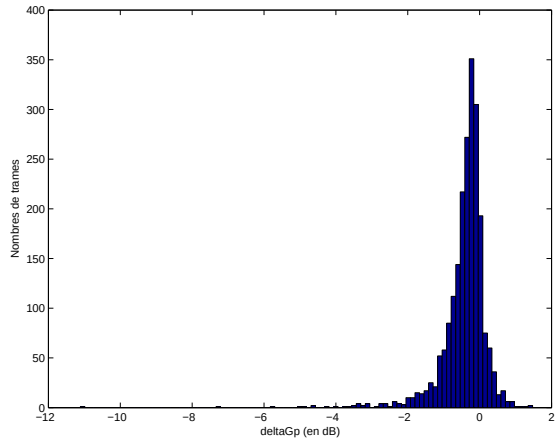


Fig. 2.13 - Histogramme de ΔG_p pour la voie gauche de clapton

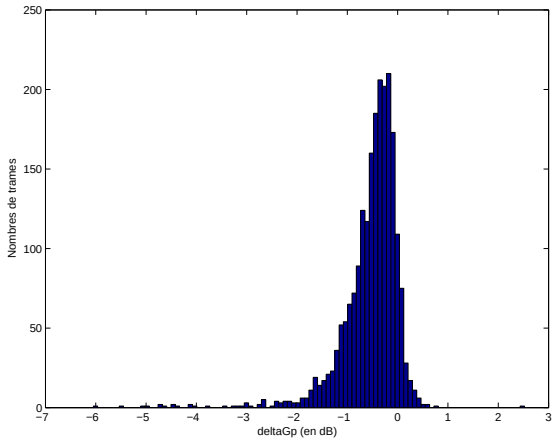


Fig. 2.14 - Histogramme de ΔG_p pour la voie droite de clapton

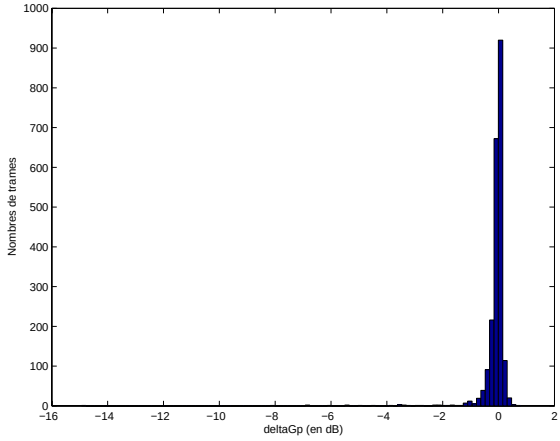


Fig. 2.15 - Histogramme de ΔG_p pour la voie gauche de vega

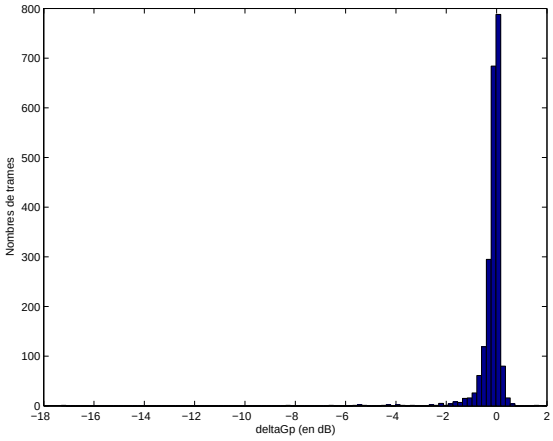


Fig. 2.16 - Histogramme de ΔG_p pour la voie droite de vega

On remarque qu'il y a tout de même certaines trames dont le gain est positif, ce gain pouvant atteindre 2 dB, c'est pourquoi la technique qui consiste à choisir, pour chaque trame, la prédiction qui donne le meilleur gain de prédiction peut améliorer les résultats.

Bilan

Si on fait un bilan sur tous les fichiers, on obtient en moyenne un gain de -0.01dB pour la voie gauche et -0.32dB sur la voie droite. Lorsqu'on choisit la meilleure prédiction, on obtient en moyenne un gain de 0.34dB pour la voie gauche et 0.29dB sur la voie droite.

	canal gauche	canal droite
Gain par rapport au linéaire	-0.01dB	-0.32dB
Gain par rapport au linéaire (choix meilleur prédiction)	0.34dB	0.29dB

Tab. 2.2 - Moyenne des gains - prédiction stéréo backward

Le gain n'est donc pas trés important en moyenne et si on considère la grande complexité de calcul que requiert la prédiction non linéaire, cette technique est au final peu intéressante.

2.3 Prédiction non linéaire monophonique sur les résidus de la prédiction linéaire stéréo avec un filtre de Volterra

On peut envisager une autre configuration : séparer la prédiction linéaire et la prédiction non linéaire en deux étapes distinctes : on effectue d'abord une prédiction linéaire stéréo puis on applique une prédiction non linéaire monophonique à chaque résidu de la prédiction linéaire.

2.3.1 Prédiction non linéaire avec un filtre de Volterra : approche backward

Comme on l'a vu précédemment, il n'y a aucun intérêt à réaliser une prédiction forward étant donné le nombre très important de coefficients à transmettre. Nous allons donc étudier dans ce qui suit une prédiction backward sur chaque résidu de la prédiction linéaire stéréo.

$$\begin{aligned}\widehat{r}_1(n) &= \sum_{i=1}^{P} h_1(i) r_1(n-i) + \sum_{i=1}^{P}\sum_{j=i}^{P} h_2(i,j).r_1(n-i)r_1(n-j) \\ \widehat{r}_2(n) &= \sum_{i=1}^{P} h_1(i) r_2(n-i) + \sum_{i=1}^{P}\sum_{j=i}^{P} h_2(i,j).r_2(n-i)r_2(n-j)\end{aligned}$$

où r_1 et r_2 sont les résidus de la prédiction linéaire stéréo.

Mise en oeuvre :

La longueur des trames est de 20ms, le recouvrement de 5ms. On pondère la trame étendue par une fenêtre asymétrique afin de privilégier les échantillons les plus récents : 95% de la fenêtre correspond à la moitié d'une fenêtre de Hamming et la partie restante est un quart de période d'une fonction cosinus. Pour que le prédicteur soit plus adaptatif, on décide d'effectuer une analyse en sous-trames : le prédicteur est mis à jour 4 fois par trames, le prédicteur restant identique sur des sous-trames de 5ms.

Le tableau suivant (figure 2.16) montre le gain par rapport à une prédiction linéaire. Le prédicteur linéaire utilise le même algorithme mais sans la partie non-linéaire. On a choisit un ordre $P = 10$.

		Δ Gp min	Δ Gp max	Δ Gp moyenne	Δ Gp moyenne 2
bagp	Gauche	-0.28	1.76	0.54	0.54
	Droite	-0.05	0.41	0.05	0.05
cast	Gauche	-6.08	1.67	-0.04	0.02
	Droite	-5.56	1.13	-0.03	0.01
clapton	Gauche	-6.36	1.46	-0.13	0.03
	Droite	-5.11	0.40	-0.12	0.00
engl	Gauche	-1.03	0.38	-0.00	0.01
	Droite	-0.01	0.02	0.00	0.00
engl8	Gauche	-12.58	4.17	-0.04	0.17
	Droite	-0.07	0.06	0.00	0.00
germ	Gauche	-1.40	1.01	0.02	0.04
	Droite	-0.00	0.04	0.00	0.00
germ8	Gauche	-10.11	5.69	0.14	0.50
	Droite	-0.00	0.00	0.00	0.00
gloc	Gauche	-1.05	1.16	0.00	0.00
	Droite	-2.77	1.31	0.00	0.00
harp	Gauche	-3.06	0.40	-0.01	0.01
	Droite	-1.28	0.46	-0.01	0.01
orch	Gauche	-0.20	0.41	0.00	0.01
	Droite	-0.09	0.37	0.01	0.01
pitc	Gauche	-0.00	2.39	0.26	0.26
	Droite	-0.00	1.02	0.18	0.18
popm	Gauche	-0.92	0.70	0.00	0.04
	Droite	-2.60	1.12	-0.05	0.02
stri	Gauche	-1.70	0.30	-0.00	0.00
	Droite	-3.83	0.26	-0.02	0.00
trum	Gauche	-0.09	0.19	0.00	0.00
	Droite	-0.03	0.15	0.00	0.00
u2_court	Gauche	-7.89	0.78	-0.33	0.00
	Droite	-3.23	0.62	-0.26	0.00
vega	Gauche	-2.25	0.48	0.00	0.01
	Droite	-0.09	0.08	0.00	0.00

Fig. 2.17 - Différence entre les gains de prédiction obtenus avec une prédiction non linéaire et les gains de prédiction obtenus avec une prédiction linéaire ($\Delta > 0 \Longrightarrow G_{Pnonlin} > G_{Plin}$) pour une prédiction mono backward sur les résidus

On remarque que le gain est négatif ou très proche de 0 en moyenne sur chaque trame sauf pour deux fichiers : bagp et pitc, et pour le canal gauche de germ8.

Bilan

Si on fait un bilan sur tous les fichiers, on obtient en moyenne un gain de 0.03dB pour la voie gauche et -0.02dB sur la voie droite. Lorsqu'on choisit la meilleure prédiction, on obtient en moyenne un gain de 0.10dB pour la voie gauche et 0.02dB sur la voie droite. Le gain est pratiquement nul. Cette technique ne présente donc pas d'intérêt.

	canal gauche	canal droite
Gain par rapport au linéaire	0.03dB	-0.02dB
Gain par rapport au linéaire (choix meilleur prédiction)	0.10dB	0.02dB

Tab. 2.3 - Moyenne des gains - prédiction mono backward sur résidus

2.3.2 Prédiction avec un filtre de Volterra adaptatif

Ces filtres, décrits dans [17], utilisent des algorithmes pour mettre à jour leurs coefficients pour chaque nouvel échantillon du signal d'entrée. Toutes les démonstrations sont aussi présentées en détail dans [41].

Le filtre LMS

C'est le filtre le plus simple qui minimise l'erreur quadratique

$$e\left(n\right)^2 = \left(x\left(n\right) - \widehat{x}\left(n\right)\right)^2 \tag{2.30}$$

avec

$$\widehat{x}\left(n\right) = \sum_{i=1}^{P} h_1(i,n)x(n-i) + \sum_{i=1}^{P}\sum_{j=i}^{P} h_2(i,j,n).x(n-i)x(n-j)$$

Les équations de mise à jour des coefficients sont

$$\begin{aligned} h_1\left(i,n+1\right) &= h_1\left(i,n\right) - \frac{\mu_1}{2}\frac{\partial e\left(n\right)}{\partial h_1(i,n)} &(2.31)\\ &= h_1\left(i,n\right) + \mu_1 e\left(n\right)x(n-i) &(2.32)\end{aligned}$$

et

$$\begin{aligned} h_2\left(i,j,n+1\right) &= h_2\left(i,j,n\right) - \frac{\mu_2}{2}\frac{\partial e\left(n\right)}{\partial h_2(i,j,n)} &(2.33)\\ &= h_2\left(i,j,n\right) + \mu_2 e\left(n\right)x(n-i)x(n-j) &(2.34)\end{aligned}$$

où μ_1 et μ_2 sont des constantes positives.

Avec les notations vectorielles :

$$\begin{aligned} e(n) &= x(n) - \mathbf{h}_n^T\mathbf{x}_n &(2.35)\\ \mathbf{h}_{n+1} &= \mathbf{h}_n + \boldsymbol{\mu}\mathbf{x}_n e(n) &(2.36)\end{aligned}$$

où $\boldsymbol{\mu}$ est une matrice diagonale dont les P premiers coefficients sont μ_1 et dont les autres sont μ_2.

On pourra donc utiliser l'algorithme suivant :

Algorithme LMS :

- Initialisation

  $\mathbf{h}_0$ est initialisé à 0.
- Boucle

  $\widehat{x}\left(n\right) = \mathbf{h}_n^T\mathbf{x}_n$

  $e(n) = x(n) - \widehat{x}\left(n\right)$

  $\mathbf{h}_{n+1}$=$\mathbf{h}_n + \boldsymbol{\mu}\mathbf{x}_n e(n)$

Cependant, au filtre LMS, on préférera le filtre NLMS qui est moins sensible aux variations de dynamique de l'entrée et a de meilleurs propriétés de convergence.

Le filtre NLMS

Le filtre NLMS est un filtre LMS normalisé. L'équation de mise à jour des coefficients est :

$$\mathbf{h}_{n+1}=\mathbf{h}_n + \left[\mu_1 \frac{\mathbf{x}_{1n}}{\delta + \|\mathbf{x}_{1n}\|} + \mu_2 \frac{\mathbf{x}_{2n}}{\delta + \|\mathbf{x}_{2n}\|}\right] e(n)$$

où δ est une petite constante qui empêche une division par 0. $\mathbf{x}_{1n}$ est la partie linéaire de $\mathbf{x}_n$ et $\mathbf{x}_{2n}$ la partie non linéaire.

Mise en oeuvre (les tests sont effectués sur 2 fichiers : clapton et vega) :

1. On applique d'abord le filtre sans la partie non linéaire afin de déterminer le $\boldsymbol{\mu}_1$ optimal (celui qui maximise la moyenne des gains de prédiction par trame).

   Fichier Clapton :

ordre	5	10
mu1 opt.	0.03	0.06
Gain voie gauche	0.83	1.00
Gain voie droite	1.47	1.66

Tab. 2.4 - Gains de prédiction NLMS - fichier clapton

   Fichier Vega :

ordre	5	10
mu1 opt.	0.03	0.06
Gain voie gauche	0.93	1.46
Gain voie droite	1.34	1.86

Tab. 2.5 - Gains de prédiction NLMS - fichier vega

   On peut déja remarquer que la prédiction NLMS linéaire donne de meilleurs gains de prédictions qu'une prédiction linéaire backward pour un ordre 10 (voir 2.3.1) :

	NLMS	linéaire backward
clapton voie gauche	1.00	0.78
clapton voie droite	1.66	1.54
clapton voie gauche	1.46	1.53
clapton voie droite	1.86	1.81

Tab. 2.6 - Gains de prédiction - comparaison NLMS avec linéaire backward

   De plus, on peut augmenter l'ordre du NLMS sans trop augmenter la complexité des calculs contrairement à la prédiction linéaire backward.

2. On applique le filtre avec la partie non linéaire : pour le $\boldsymbol{\mu}_1$ déterminé précédemment, on recherche $\boldsymbol{\mu}_2$ qui maximise le gain de prédiction. Malheuresement, quelque soit l'ordre et quelque soit $\boldsymbol{\mu}_2$, le gain de prédiction est inférieure en moyenne au gain obtenu avec une prédiction

linéaire. La figure suivante montre la moyenne des gains de prédiction par trame en fonction de $\boldsymbol{\mu}_2$ pour les voi

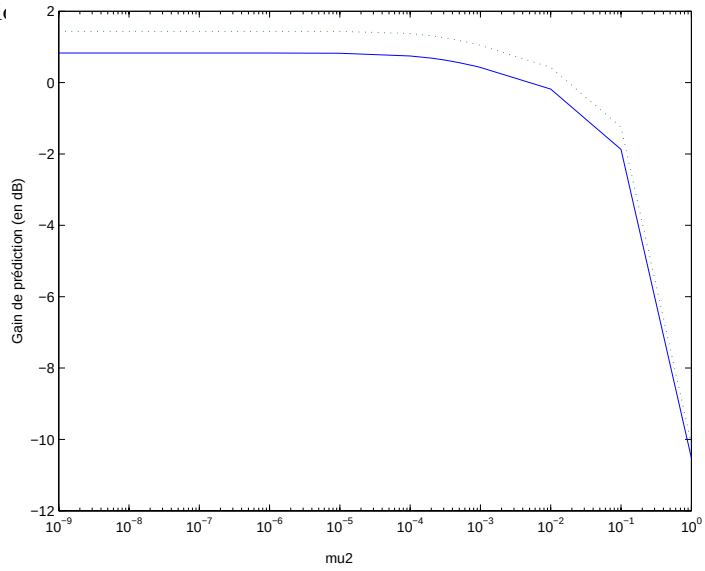


Fig 2.17 - Gain de prédiction en fonction de mu2

3. Par contre, pour certaines trames, la prédiction non linéaire donne un meilleur gain de prédiction. On pourrait donc envisager une technique qui consisterait, pour chaque trame, à choisir la prédiction qui donne le meilleur gain de prédiction. La figure suivante montre la moyenne des gains de prédiction par trame obtenue avec cette technique, en fonction de $\boldsymbol{\mu}_2$, pour les voies gauche et droite de clapton et pour un ordre 5.

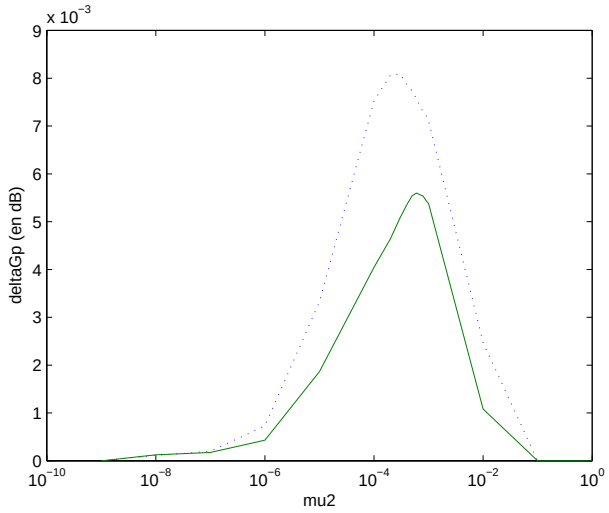


Fig. 2.18 - Gain par rapport au linéaire en fonction de mu2 - choix de la meilleure prédiction

On atteint un maximum pour $10^{-4} < \boldsymbol{\mu}_2 < 10^{-3}$. Mais le gain apporté est tellement faible (au maximum 8.10^{-3} dB) que cette technique présente peu d'intérêt.

Le filtre AP (Affine projection)

Algorithme AP :

– Initialisation

$\mathbf{h}_0$ est initialisé à 0.

– Boucle

$\mathbf{R} = [\mathbf{x}_n \ \mathbf{x}_{n-1} \ ... \ \mathbf{x}_{n-q+1}]$

$$\mathbf{e} = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-q+1) \end{bmatrix} - \mathbf{R}^T \mathbf{h}_{n-1}$$

$$\mathbf{h}_n = \mathbf{h}_{n-1} + \mu \mathbf{R} \left\{ \mathbf{R}^T \mathbf{R} \right\}^{-1} \mathbf{e}$$

Mise en oeuvre (les tests sont effectués sur 2 fichiers : clapton et vega) :

1. On applique d'abord le filtre sans la partie non linéaire. L'ordre de projection est 2.

   Fichier Clapton :

ordre	5	10
mu opt.	0.02	0.03
Gain voie gauche	0.72	1.92
Gain voie droite	1.27	1.51

Tab. 2.7 - Gains de prédiction AP - fichier clapton

   Fichier Vega :

ordre	5	10
mu opt.	0.02	0.03
Gain voie gauche	0.82	1.39
Gain voie droite	1.20	1.74

Tab. 2.8 - Gains de prédiction AP - fichier vega

   On remarque tout d'abord que les gains de prédiction sont inférieurs à ceux obtenus avec le filtre NLMS.

2. On applique ensuite le filtre avec la partie non linéaire, mais comme pour le filtre NLMS, la partie non linéaire n'améliore pas le gain de prédiction.

Le filtre RLS

Ce filtre minimise la fonction d'erreur

$$J(n) = \sum_{k=0}^{n} \lambda^{n-k} \left(x(k) - \mathbf{h}_n^T \mathbf{x}_k\right)^2 \tag{2.37}$$

où $0 \preceq \lambda \preceq 1$,λ est un facteur qui controle la capacité de mémoire du filtre.
En annulant la dérivé de J, on trouve facilement les coefficients optimaux :

$$\mathbf{h}_n = \mathbf{C}^{-1}(n)\mathbf{p}(n) \tag{2.38}$$

avec

$$\mathbf{C}(n) = \sum_{k=0}^{n} \lambda^{n-k} \mathbf{x}_k \mathbf{x}_k^T \tag{2.39}$$

et

$$\mathbf{p}(n) = \sum_{k=0}^{n} \lambda^{n-k} x(k) \mathbf{x}_k \tag{2.40}$$

On peut tout d'abord déduire de (2.39) et (2.40) :

$$\mathbf{C}(n) = \lambda \mathbf{C}(n-1) + \mathbf{x}_n \mathbf{x}_n^T \tag{2.41}$$

$$\mathbf{p}(n) = \lambda \mathbf{p}(n-1) + x(n)\mathbf{x}_n \tag{2.42}$$

En utilisant le lemme d'inversion matricielle, on déduit alors de (2.41) :

$$\mathbf{C}^{-1}(n) = \lambda^{-1}\mathbf{C}^{-1}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{x}_n^T \mathbf{C}^{-1}(n-1) \tag{2.43}$$

avec

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{C}^{-1}(n-1)\mathbf{x}_n}{1 + \lambda^{-1}\mathbf{x}_n^T \mathbf{C}^{-1}(n-1)\mathbf{x}_n} \tag{2.44}$$

Mais 2.44 est aussi équivalent à

$$\begin{aligned} \mathbf{k}(n) &= \lambda^{-1}\mathbf{C}^{-1}(n-1)\mathbf{x}_n - \lambda^{-1}\mathbf{k}(n)\mathbf{x}_n^T \mathbf{C}^{-1}(n-1)\mathbf{x}_n \\ &= \left[\lambda^{-1}\mathbf{C}^{-1}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{x}_n^T \mathbf{C}^{-1}(n-1)\right]\mathbf{x}_n \\ &= \mathbf{C}^{-1}(n)\mathbf{x}_n \end{aligned} \tag{2.45}$$

En utilisant (2.41) , (2.42) et (2.45), on a finalement :

$$\begin{aligned} \mathbf{h}_n &= \mathbf{C}^{-1}(n)\mathbf{p}(n) \\ &= \lambda \mathbf{C}^{-1}(n)\mathbf{p}(n-1) + \mathbf{C}^{-1}(n)\mathbf{x}_n x(n) \\ &= \mathbf{C}^{-1}(n-1)\mathbf{p}(n-1) - \mathbf{k}(n)\mathbf{x}_n^T \mathbf{C}^{-1}(n-1)\mathbf{p}(n-1) + \mathbf{C}^{-1}(n)\mathbf{x}_n x(n) \\ &= \mathbf{h}_{n-1} - \mathbf{k}(n)\mathbf{x}_n^T \mathbf{h}_{n-1} + \mathbf{k}(n)x(n) \\ &= \mathbf{h}_{n-1} + \mathbf{k}(n)e(n) \qquad \text{avec } e(n) = x(n) - \mathbf{h}_{n-1}^T \mathbf{x}_n \end{aligned}$$

On pourra donc utiliser l'algorithme suivant :

Algorithme RLS :

– Initialisation

$\mathbf{h}_0$ est initialisé à 0.

$\mathbf{C}^{-1}(0) = \delta^{-1}\mathbf{I}$ avec δ une petite constante positive

– Boucle

$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{C}^{-1}(n-1)\mathbf{x}_n}{1+\lambda^{-1}\mathbf{x}_n^T\mathbf{C}^{-1}(n-1)\mathbf{x}}$

$\widehat{x}\,(n) = \mathbf{h}_{n-1}^T\mathbf{x}_n$

$e(n) = x(n) - \widehat{x}\,(n)$

$\mathbf{h}_n = \mathbf{h}_{n-1} + \mathbf{k}(n)e(n)$

$\mathbf{C}^{-1}(n) = \lambda^{-1}\mathbf{C}^{-1}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{x}_n^T\mathbf{C}^{-1}(n-1)$

Mise en oeuvre (les tests sont effectués sur 2 fichiers : clapton et vega). On prend $\lambda = 1$, $\delta = 0.00001$:

Fichier Clapton :

ordre	5		10	
	Lin	Non Lin	Lin	Non Lin
Gain voie gauche	0.02	0.04	0.03	-0.17
Gain voie droite	-0.08	0.00	-0.04	0.04

Tab. 2.9 - Gains de prédiction RLS - fichier clapton

Fichier Vega :

ordre	5		10	
	Lin	Non Lin	Lin	Non Lin
Gain voie gauche	0.07	0.06	0.08	-0.01
Gain voie droite	0.08	0.04	0.09	0.07

Tab. 2.10 - Gains de prédiction RLS - fichier vega

On remarque que les gains de prédiction sont inférieurs à ceux obtenus avec les 2 filtres précédents. Il n'y a donc pas d'intérêt à utiliser cette technique même si le non linéaire améliore le gain dans certains cas.

2.4 Conclusions

Le but de ce chapitre était de voir si une prédiction basée sur un filtre de volterra donnait de meilleurs gains de prédiction qu'une simple prédiction linéaire. Nous avons d'abord étudié une prédiction stéréo adaptative par trame (approche forward et backward) puis nous avons étudié deux types de prédiction mono sur les résidus de la prédiction linéaire stéréo : une prédiction adaptative par trame (approche backward) et une prédiction adaptative par échantillon. Nous sommes parvenu aux conclusions suivantes :

– Un prédicteur stéréo basée sur un filtre de Volterra donne de meilleurs gains de prédiction qu'une prédiction stéréo linéaire en forward. Mais ce type de prédiction coüte trop cher en

terme de débit à cause du nombre très important de coefficients à transmettre.

- En backward, un prédicteur stéréo basé sur un filtre de Volterra améliore trés peu les gains de prédiction d'une prédiction stéréo linéaire au prix d'une grande complexité.
- Une prédiction mono sur les résidus avec un filtre de Volterra n'améliore pas ou alors trés peu les gains de prédiction d'une prédiction linéaire, que ce soit pour un prédicteur adaptatif par trame ou un prédicteur adaptatif par échantillon.

Finalement, nous pouvons en conclure que la prédiction non linéaire avec un filtre de Volterra, quelle que soit l'approche choisie, n'améliore pas les performances d'un codeur audio sans perte.

Chapitre 3

Prédiction non linéaire avec un réseau de neurones

Dans [13], les auteurs montrent qu'un réseau de neurones est plus performant qu'un filtre de Volterra pour prédire des signaux de parole (en forward). De plus un réseau de neurones utilise moins de coefficients qu'un filtre de Volterra pour des ordres égaux. Les résultats prometteurs de cet article nous ont donc poussé à étudier ce type de prédiction dans le cas de signaux audio. Nous verrons donc dans cette partie si une prédiction avec un réseau de neurone est plus performante qu'une prédiction linéaire pour un codeur audio sans perte.

Nous rappellerons dans un premier temps ce qu'est un réseau de neurones. Puis nous étudierons, comme nous l'avons fait pour le filtre de Volterra, une approche forward et une approche backward du prédicteur.

3.1 Généralités sur les réseaux de neurones

3.1.1 Le neurone formel

La description d'un réseau de neurones commence par la représentation d'un neurone formel. Le modèle le plus simple et le plus ancien de neurone formel est le modèle Mc Culloch et Pitts [19]. Le "neurone" de Mc Culloch et Pitts est un automate booléen à seuil dont la sortie S est donnée par la fonction de transfert :

$$\begin{cases} S = f(x - \theta) \\ \text{où} \\ x = \sum_{i=1}^{n} w_i.e_i \end{cases} \tag{3.1}$$

– Le vecteur $(e_i)_{i=1..n}$ représente le vecteur des entrées de l'unité.

– Le vecteur $(w_i)_{i=1..n}$ est le vecteur des "poids" des connexions associées à l'unité. Le poids w_i pondère l'influence de la i-ème entrée dans le signal total reçu par l'unité.
– θ est un paramètre de seuil.
– x est l'état d'activation de l'unité dont l'analogue biologique est le potentiel membranaire.

Dans le modèle d'origine, la fonction f est une fonction "seuil" qui peut être indifféremment, une fonction de Heaviside :

$$f(x) = \begin{cases} 1 \text{ si } x \succeq 0 \\ 0 \text{ si } x \prec 0 \end{cases} \tag{3.2}$$

ou,

une fonction "signe" :

$$f(x) = \begin{cases} +1 \text{ si } x \succeq 0 \\ -1 \text{ si } x \prec 0 \end{cases} \tag{3.3}$$

Les biologistes et les chercheurs soucieux de vraisemblance biologique préfèrent donner une description continue du neurone. La fonction f est alors une fonction bornée, continue et croissante. Les fonctions les plus courantes sont les fonctions "sigmoïdes" :

$$f(x) = \frac{1}{1 + \exp(-\beta x)} \tag{3.4}$$

et

$$f(x) = \frac{1 - \exp(-\beta x)}{1 + \exp(-\beta x)} \tag{3.5}$$

qui sont respectivement a valeurs dans $[0, 1]$ et dans $[-1, 1]$. Le paramètre β est un paramètre de gain qui mesure la raideur de la sigmoïde. Remarquons que dans la limite d'un gain infini, on retrouve les fonctions a valeurs discrètes données précédemment.

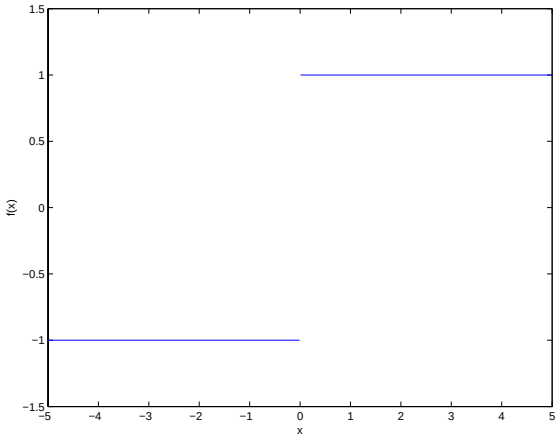


Fig. 3.1 - Fonction "seuil"

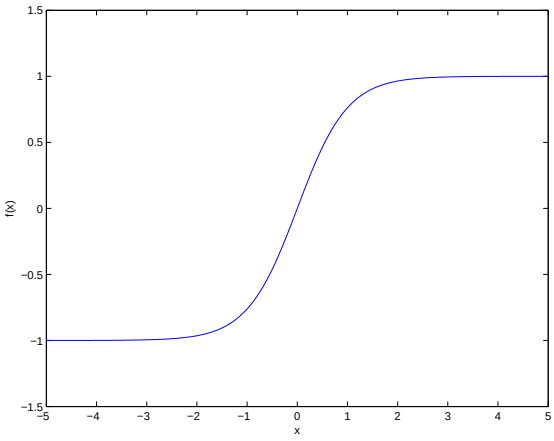


Fig. 3.2 : Fonction "sigmoïde"

Dans toute la suite, le neurone formel aura comme fonction de transfert une fonction sigmoïde à valeurs dans $[-1, 1]$ et un gain $\beta = 1$. De plus, on fixera le seuil θ à 0 et on ajoutera une constante

b appellé biais. La relation entrée-sortie du neurone est alors donnée par les égalités :

$$\begin{cases} S = \frac{1-\exp(-x)}{1+\exp(-x)} + b \\ \text{où} \\ x = \sum_{i=1}^{n} w_i.e_i \end{cases} \tag{3.6}$$

3.1.2 Le réseau de neurones

L'interconnexion d'un ensemble d'unités du type de celles décrites dans le paragraphe précédent définit un réseau de neurones. On peut distinguer deux grandes familles de réseaux : les réseaux non-récurrents ou « feedforward » et les réseaux récurrents.

Dans un réseau « feedforward », les unités sont organisées en couches successives. Les sorties des unités d'une couche servent d'entrées aux unités de la couche suivante. Au sein d'une même couche, les unités ne sont pas connectées entre elles. La première couche est appelée la couche d'entrée, la dernière, la couche de sortie et les couches intermédiaires sont appelées les couches cachées.

S'il existe des boucles, dans le graphe orienté des connexions, le réseau est dit récurrent.

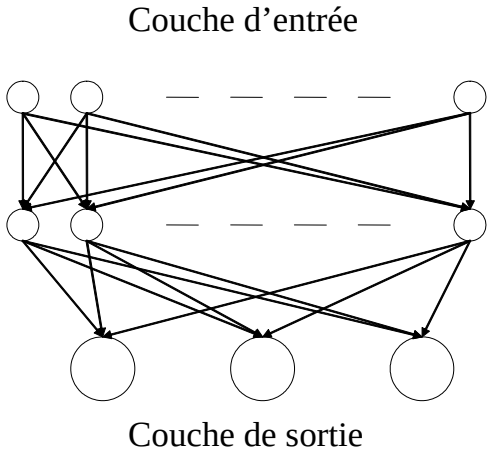


Fig. 3.3 - Réseau "feedforward"

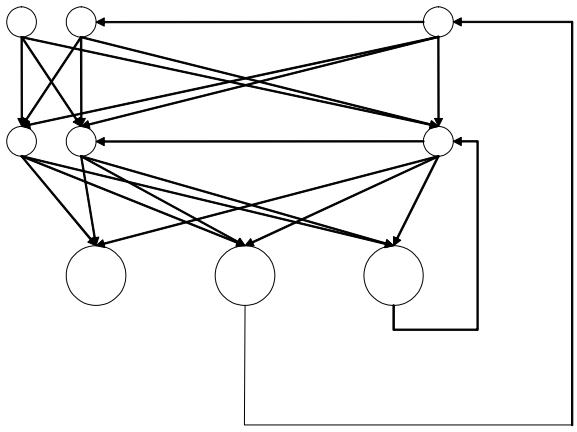


Fig. 3.4 - Réseau récurrent

3.1.3 Le Perceptron - Historique (extraits de [24])

C'est en 1958 que Rosenblatt [20] conçoit le Perceptron, un réseau de neurones "feedforward" possédant seulement deux couches : une couche d'entrée et une couche de sortie. Ce réseau, qui parvient à apprendre à identifier des formes simples et à calculer certaines fonctions logiques, constitue le premier système artificiel exhibant une faculté jusque là réservée au vivant : la capacité d'apprendre par l'expérience ; c'est le premier réseau de neurones artificiel proprement dit.

Les travaux de Rosenblatt suscitent au début des années 60 un vif enthousiasme chez les scientifiques alors fortement impliqués dans la recherche sur l'intelligence artificielle. Cet enthousiasme se

voit pourtant brusquement refroidi en 1969 lorsque deux scientifiques américains de renom, Minsky et Papert [21], publient un livre qui met à jour les limites intrinsèques du Perceptron.

Ce qu'ont démontré Minsky et Papert c'est qu'un réseau de neurones de type perceptron est incapable de résoudre toute une classe de problèmes simples (les problèmes non linéairement séparables). Certes l'utilisation de couches intermédiaires, "cachées", de neurones, permettrait de contourner cette limitation, à condition de disposer d'un mécanisme d'apprentissage approprié pour ces neurones additionnels. Mais c'est précisément ce mécanisme qui à l'époque fait cruellement défaut, ce qui fait dire en substance aux deux savants américains, qu' "un réseau de type perceptron ne sera jamais capable de faire quoi que ce soit d'intéressant".

Il faut attendre le début des années 80 pour voir un regain d'intérêt pour les réseaux de neurones artificiels. Celui-ci s'explique tout d'abord par les résultats des travaux de Hopfield [22] qui démontre, en 1982, l'utilité des réseaux récurrents dans la compréhension et la modélisation des processus de la mémoire et rend manifeste la relation existant, sur le plan formel, entre ce type de réseaux et des systèmes physiques (tels que les verres de spin) pour lesquels la physique statistique fournit un cadre théorique parfaitement approprié. Parallèlement aux travaux de Hopfield, Werbos conçoit un mécanisme d'apprentissage pour les réseaux multicouches de type perceptron : c'est l'algorithme d'apprentissage par "Back-propagation" (rétro-propagation de l'erreur) qui fournit un moyen simple d'entraîner les neurones des couches cachées. Cet algorithme sera réellement popularisé en 1986 par Rumelhart et al [23] dans un livre : "Parallel Distributed Processing".

3.1.4 L'apprentissage

Les techniques d'apprentissage se répartissent en trois grandes familles :

- L'apprentissage non supervisé modifie les poids du réseau en fonction d'un critère interne, indépendant de l'adéquation entre le comportement du réseau et la tache qu'il doit effectuer.
- L'apprentissage supervisé, au contraire, dispose d'un comportement de référence vers lequel il tente de faire converger le réseau.
- L'apprentissage semi-supervisé suppose qu'un comportement de référence précis n'est pas disponible, mais qu'en revanche il est possible d'obtenir des indications qualitatives ou lacunaires sur les performances du réseau.

Nous nous intéresserons ici uniquement à l'apprentissage supervisé.

Le but de l'apprentissage supervisé est d'inculquer un comportement de référence au réseau. Ce comportement de référence est défini à partir d'un ensemble représentatif d'exemples appelé base d'apprentissage. Cet ensemble est composé de couples d'apprentissage formés d'un vecteur d'entrée appelé patron d'entrée et d'un vecteur de la sortie désirée appelé patron de référence.

L'apprentissage consistera alors à déterminer les poids des différentes liaisons du réseau de manière à obtenir, lorsqu'on présente au réseau un patron d'entrée, une valeur en sortie aussi proche que possible du patron de référence correspondant.

L'apprentissage comprend quatre étapes de calcul :

1. Initialisation des poids du réseau.
2. Présentation des patrons d'entrée de la base d'apprentissage et calcul de l'état du réseau.
3. Calcul de l'erreur. L'erreur est une fonction de la différence entre la sortie obtenue et la sortie désirée. On choisira en général l'erreur quadratique.
4. Modification des poids synaptiques. Cette modification s'effectue grâce à un algorithme d'apprentissage comme par exemple la rétro-propagation du gradient.

Les étapes 2-3-4 (une époque) sont répétées jusqu'à la fin de l'apprentissage : on s'arrête lorsque l'erreur descend en dessous d'une certaine valeur ou après un certain nombre d'époques.

3.1.5 Réseaux de neurones pour la prédiction non linéaire

Dans la littérature, trois types de réseaux de neurones ont été utilisés pour la prédiction non linéaire :

- Les réseaux de type perceptron multi-couche [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35].
- Les réseaux à fonctions de bases radiales [36], [37], [39], [40].
- Les réseaux récurrents [42], [43].

Quelque soit le réseau utilisé, on cherchera à prédire un signal $x(n)$ à partir de ses P valeurs précédentes. Le réseau aura donc P entrées et 1 sortie et il sera entrainé de telle manière qu'il ait en sortie une valeur aussi proche que possible de $x(n)$ lorsqu'on lui présente en entrée les valeurs $x(n-1), x(n-2), ..., x(n-P)$.

On étudiera une prédiction non linéaire avec le réseau de neurones dont la structure est la plus simple : le perceptron multi-couches ou multi-layer perceptron (MLP). Marcos Faundez a étudié ce type de prédiction pour un codeur de parole de type ADPCM [25]-[31]. Une grande partie de mon étude s'est inspirée de ses travaux.

3.2 Prédiction non linéaire stéréo avec un réseau MLP : approche forward

On étudie dans un premier temps une approche forward du prédicteur.

3.2.1 Les réseaux utilisés

Prédiction du canal gauche

Le premier réseau prédit $x_1(n)$ à partir des échantillons passés $x_1(n-i)$ et $x_2(n-j)$ avec $1 \preceq i \leq P_1$ et $1 \leq j \leq P_2$. Le premier réseau a donc $P_1 + P_2$ neurones d'entrées et une sortie.

Prédiction du canal de droite

Pour la prédiction du signal de la voie droite, on peut tirer parti de l'entrelacement des voies x_1 et x_2. Le deuxième réseau prédit donc $x_2(n)$ à partir des échantillons passés $x_1(n-i)$ et $x_2(n-j)$ avec $0 \preceq i \leq P_2$ et $1 \leq j \leq P_1$. Le deuxième réseau a donc $P_1 + P_2 + 1$ neurones d'entrées et une sortie.

Choix de P_1 et P_2

Pour comparer avec les résultats obtenus avec une prédiction linéaire stéréo et avec une prédiction stéréo avec un filtre de Volterra, on choisira $P_1 = 20$ et $P_2 = 10$.

Choix du nombre de neurones cachées

Plus le réseau MLP a de neurones, plus il est performant. D'un autre coté, plus le réseau a de neurones, plus l'apprentissage nécessitera des calculs complexes et plus le prédicteur aura de paramètres et donc plus le codeur aura de paramètres à transmettre. Dans [25], les auteurs ont utilisés un réseau MLP pour prédire des signaux de parole. Il a été montré qu'une couche cachée avec deux neurones cachés était un bon compromis entre la performance et la complexité des calculs. Bien que les signaux audio que nous allons utiliser soient plus riches que des signaux de parole, nous allons tout de même utiliser des réseaux à deux neurones cachés pour limiter le nombre de paramètres.

3.2.2 Apprentissage

Algorithme d'Apprentissage

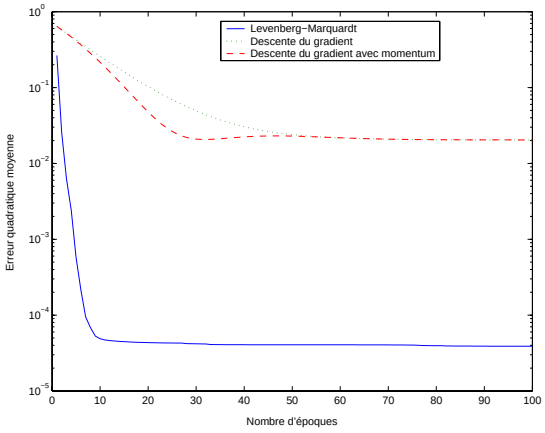


Fig. 3.5 - Erreur quadratique moyenne d'une base d'apprentissage (une trame du fichier clapton) en fonction du nombre d'époques pour différents algorithmes d'apprentissage

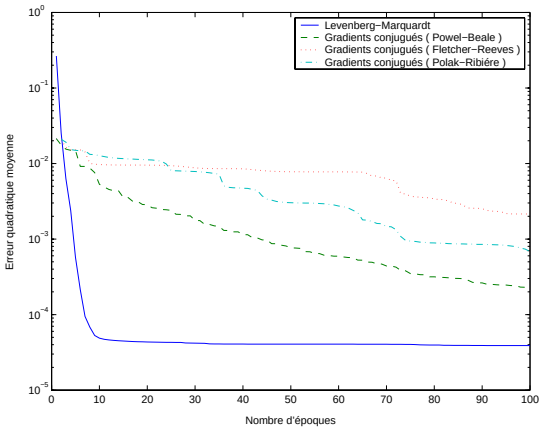


Fig. 3.6 - Erreur quadratique moyenne d'une base d'apprentissage (une trame du fichier clapton) en fonction du nombre d'époques pour différents algorithmes d'apprentissage

Pour des petits réseaux, l'algorithme de Levenberg-Marquardt [44] est le plus efficace : il converge en quelques époques mais il nécessite l'inversion d'une matrice de taille N à chaque époque (N étant le nombre de paramètres du réseau). Les figures 3.5, 3.6, 3.7 montrent l'erreur quadratique moyenne d'une base d'apprentissage (une trame du fichier clapton) en fonction du nombre d'époques pour différents algorithmes d'Apprentissage.

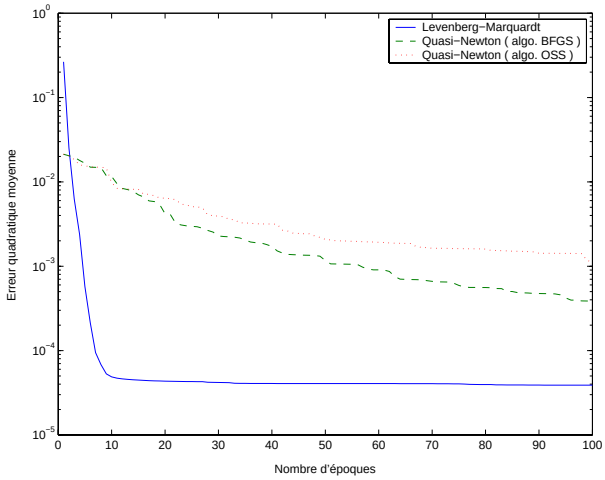


Fig. 3.7 - Erreur quadratique moyenne d'une base d'apprentissage (une trame du fichier clapton) en fonction du nombre d'époques pour différents algorithmes d'apprentissage

Le multi-start

L'article [27] propose la technique suivante : pour chaque trame, on entraîne le réseau plusieurs fois avec différentes initialisations ; on garde alors l'apprentissage qui donne le meilleur gain de prédiction. Les différentes initialisations sont :

- 1 initialisation qui reprend le réseau entraîné de la trame précédente (celui qui a donné le meilleur gain).
- 4 initialisations aléatoires : chaque poids est initialisé aléatoirement selon la règle de Nguyen-Widrow ([45]).

Cette technique est testé sur un extrait du fichier Clapton (98 trames de longueurs 1024 échantillons, l'algorithme d'apprentissage est Levenberg-Marquardt avec 50 époques). Les figures suivantes montrent le gain de prédiction pour 3 trames en fonction du nombre d'époques pour les différentes initialisations. On remarque que l'initialisation des poids du réseau a une grande influence sur la convergence de l'apprentissage :

- On peut avoir une différence de gain de 5dB à la fin de l'apprentissage pour deux initialisations différentes (figure 3.8).

– L'apprentissage avec les poids de la trame précédente converge très vite. Deux trames consécutives étant très similaires, les poids du réseau pour deux trames consécutives sont aussi assez semblables.
– L'apprentissage avec les poids de la trame précédente est dans la majorité des cas celui qui donne le meilleur gain de prédiction (environ 50% des trames).

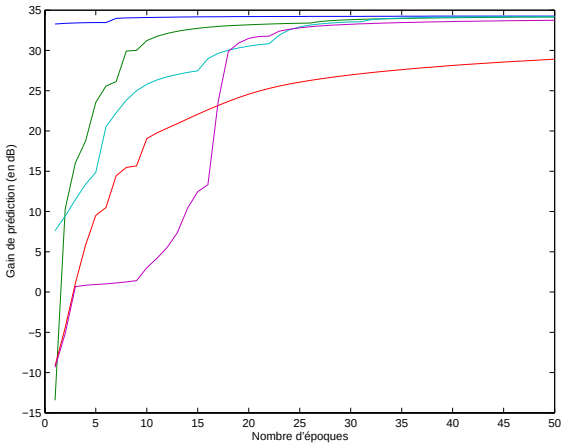


Fig. 3.8 - Gain de prédiction (en dB) en fonction du nombre d'époques pour les 5 initialisations (trame 1)

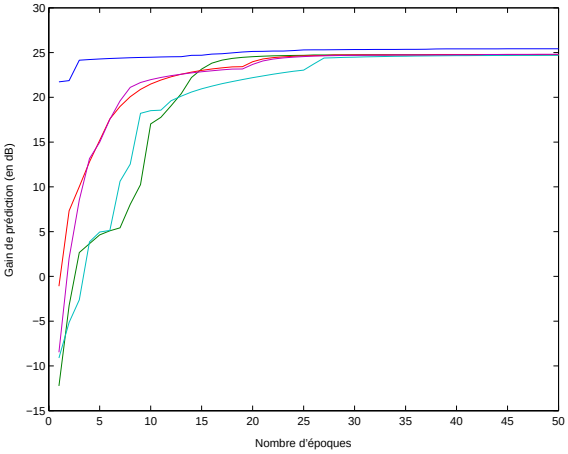


Fig. 3.9 - Gain de prédiction (en dB) en fonction du nombre d'époques pour les 5 initialisations (trame 2)

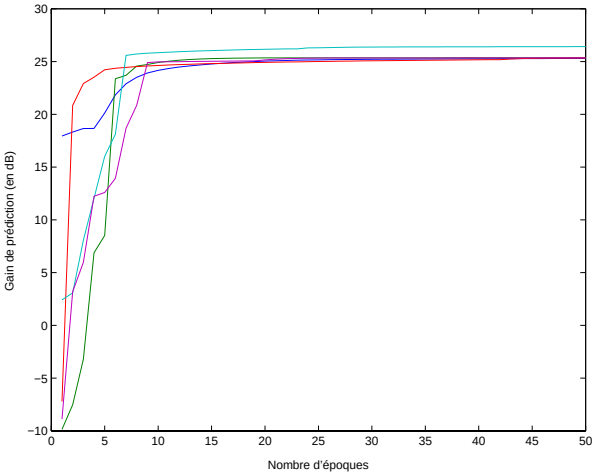


Fig. 3.10 - Gain de prédiction (en dB) en fonction du nombre d'époques pour les 5 initialisations (trame 3)

La figure suivante montre la moyenne des gains de prédiction par trame en fonction du nombre d'époques (voie gauche et voie droite de l'extrait du fichier clapton)

– avec le multi-start

– avec un seul apprentissage par trame et sans réinitialisation des poids du réseau

Le multi-start apporte donc un gain non négligeable.

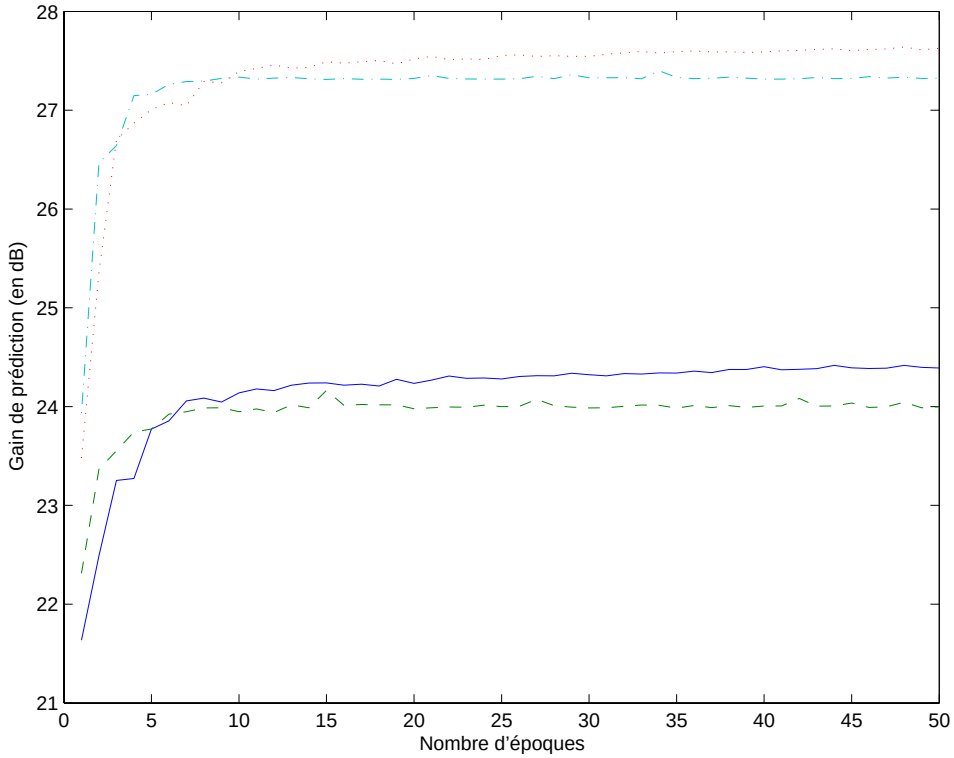


Fig. 3.11 - Gain de prédiction en fonction du nombre d'époques

avec multi-start (voie gauche : —, voie droite : · · ·) et

sans muti-start (voie gauche : - - - , voie droite : - · - ·)

3.2.3 Mise en oeuvre

L'algorithme a été testé sur les mêmes fichiers audio que pour le filtre de Volterra. On a pris une longueur de trame de 20ms. L'apprentissage utilise la technique définie précédemment (multi-start) avec 5 initialisations et 50 époques. On a alors calculé le gain de prédiction sur chaque trame des voies gauche et droite de chaque fichier.

La Figure 3.12 montre la différence avec le gain de prédiction obtenue avec un prédicteur linéaire stéréo (minimum, maximum et moyenne). La dernière colonne du tableau correspond au gain apporté par la technique qui consiste à choisir la prédiction qui donne le meilleur gain de prédiction.

		ΔGp min	*ΔGp max*	*ΔGp moyenne*	*ΔGp moyenne 2*
bagp	Gauche	-4.28	10.26	5.12	5.14
	Droite	-6.90	5.07	2.68	2.76
cast	Gauche	-.3.38	11.86	2.70	2.73
	Droite	-4.21	12.69	2.60	2.64
clapton	Gauche	0.13	6.09	1.60	1.60
	Droite	0.23	7.47	2.65	2.65
engl	Gauche	-5.16	9.83	3.11	3.19
	Droite	-13.70	30.54	10.24	10.29
engl8	Gauche	-2.52	21.03	2.76	2.86
	Droite	-4.22	23.33	5.67	5.74
germ	Gauche	-4.42	8.86	2.87	2.98
	Droite	-4.12	23.97	8.67	8.69
germ8	Gauche	-1.54	13.51	4.12	4.20
	Droite	-2.06	16.31	5.91	6.00
gloc	Gauche	-1.14	14.41	6.27	6.28
	Droite	-1.26	14.08	6.53	6.55
harp	Gauche	-0.80	8.48	2.35	2.36
	Droite	-3.17	7.13	2.49	2.50
orch	Gauche	-2.61	10.96	6.30	6.32
	Droite	-0.64	10.68	6.78	6.78
pitc	Gauche	-1.80	12.84	4.78	4.78
	Droite	-2.10	13.67	4.53	4.53
popm	Gauche	-29.62	6.78	3.88	3.95
	Droite	-4.69	6.18	3.51	3.60
stri	Gauche	-0.31	7.84	3.97	3.91
	Droite	-2.17	11.48	3.89	3.91
trum	Gauche	-8.09	13.37	8.04	8.08
	Droite	-13.68	13.74	8.26	8.31
u2_court	Gauche	-4.32	5.71	1.05	1.07
	Droite	-5.13	6.49	1.21	1.24
vega	Gauche	-6.02	6.88	2.93	2.99
	Droite	-5.30	8.91	3.17	3.20

Fig. 3.12 - Différence entre les gains de prédiction obtenus avec une prédiction non linéaire MLP et les gains de prédiction obtenus avec une prédiction linéaire ($\Delta > 0 \Longrightarrow G_{Pnonlin} > G_{Plin}$) pour une prédiction stéréo forward

Les résultats sont donc bien supérieurs à ceux obtenus avec le filtre de Volterra. Pour la grande majorité des trames, le gain par rapport au linéaire est positif. Les seules trames où le réseau se comporte moins bien qu'un prédicteur linéaire correspondent aux zones de silence des signaux.

La figure 3.13 montre une partie voisée du canal gauche du signal de parole germ (échantillonné à 48 khz) ainsi que les résidus des prédictions stéréo linéaire, non linéaire avec un filtre de Volterra et non linéaire avec un réseau MLP. La figure 3.14 montre la même partie voisée du signal mais échantillonné à 8 khz (germ8) ainsi que les résidus des prédictions. Ces figures illustrent bien le gain apporté par le réseau de neurone.

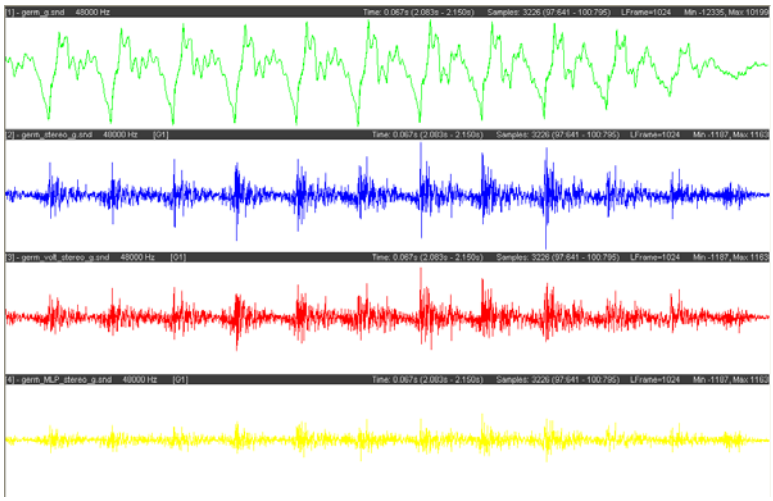


Fig. 3.13 - (de haut en bas) signal de parole échantilloné à 48 khz, résidu de la prédiction linéaire, résidu de la prédiction non linéaire (Volterra) et résidu de la prédiction non linéaire (MLP) (les résidus sont à la même échelle)

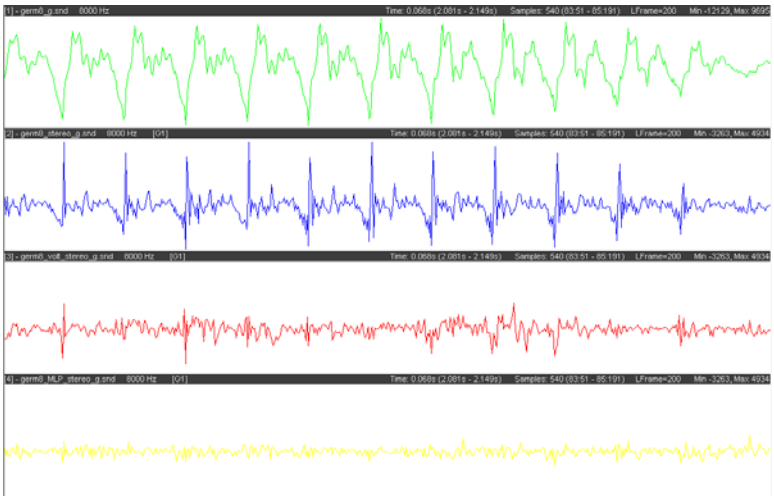


Fig. 3.14 - (de haut en bas) signal de parole échantilloné à 8 khz, résidu de la prédiction linéaire, résidu de la prédiction non linéaire (Volterra) et résidu de la prédiction non linéaire (MLP) (les résidus sont à la même échelle)

Bilan :

Si on fait un bilan sur tous les fichiers, on obtient en moyenne un gain de 3.87dB pour la voie gauche et 4.92 dB pour la voie droite. Le gain est donc beaucoup plus important que celui obtenu avec un filtre de Volterra.

	canal gauche	canal droite
Gain par rapport au linéaire	3.87dB	4.92dB
Gain par rapport au linéaire (choix meilleur prédiction)	3.90dB	4.96dB

Tab. 3.1 - Moyenne des gains - prédiction stéréo forward

3.2.4 Quantification des coefficients

Dans l'approche forward, il faut considérer la quantification des coefficients. Dans [30], un réseau MLP a été utilisé pour prédire des signaux de parole. Les paramètres du réseau ont été quantifiés avec une quantification scalaire uniforme et il a été montré que les poids et biais des différentes couches du réseau nécessitaient un nombre de bits de quantification différents : les poids de la seconde couche sont plus sensibles à la quantification que les poids de la première couche.

Les figures suivantes montrent la valeur des coefficients du réseau pour 3 trames de la voie gauche de clapton. On remarque que la dynamique des coefficients peut varier considérablement d'une trame à l'autre. On envisagera donc une quantification scalaire uniforme dont l'échelle est adaptée à chaque trame comme pour la quantification des coefficients du filtre de Volterra.

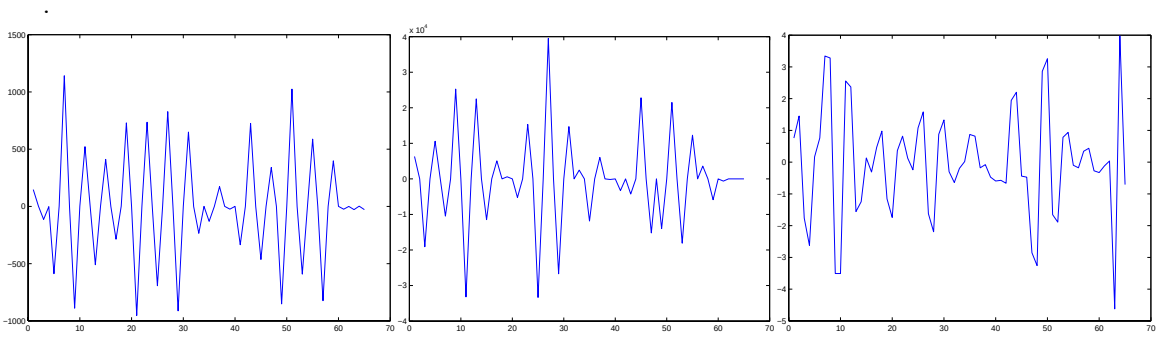


Fig. 3.15,3.16,3.17 - Valeur des coefficients du réseau (voie Gauche) pour 3 trames différentes

On a alors réalisé plusieurs tests de quantification sur les coefficients déterminés avec le fichier clapton : on a quantifié séparément les poids et les biais de la première et de la deuxième couche.

Les figures suivantes montrent le gain de prédiction obtenu sur les 2 canaux en fonction du nombre de bits de quantification. On remarque que les paramètres du réseau de neurones sont beaucoup plus sensibles à la quantification que les coefficients du filtre de Volterra (voir Fig. 2.10). Pour ne pas trop perdre en gain de prédiction, il est nécessaire, quelque soit le paramètre, de quantifier au moins sur 20 bits. Dans notre cas, on ne voit aucune différence entre les poids de la première et de la seconde couche.

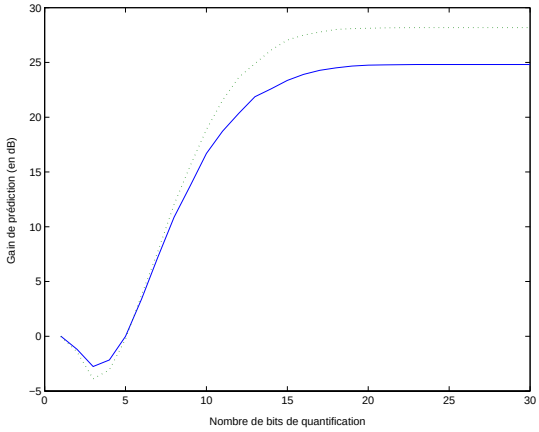


Fig. 3.18 - Gain de prédiction en fonction du nombre de bits de quantification pour les poids de la premiere couche (voie gauche : —, voie droite : - - -)

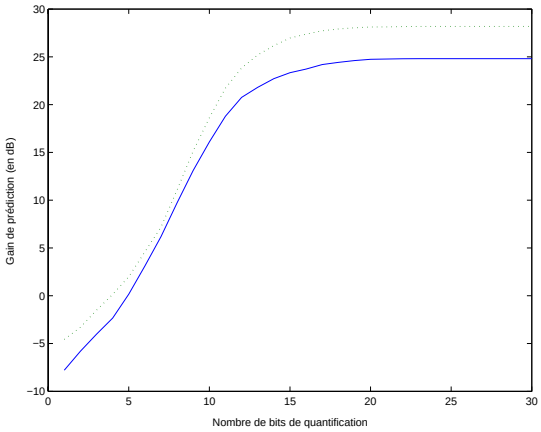


Fig. 3.19 - Gain de prédiction en fonction du nombre de bits de quantification pour les biais de la premiere couche (voie gauche : —, voie droite : - - -)

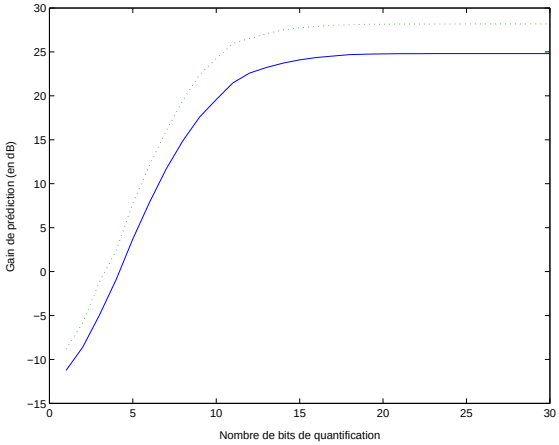


Fig. 3.20 - Gain de prédiction en fonction du nombre de bits de quantification pour les poids de la seconde couche (voie gauche : —, voie droite : - - -)

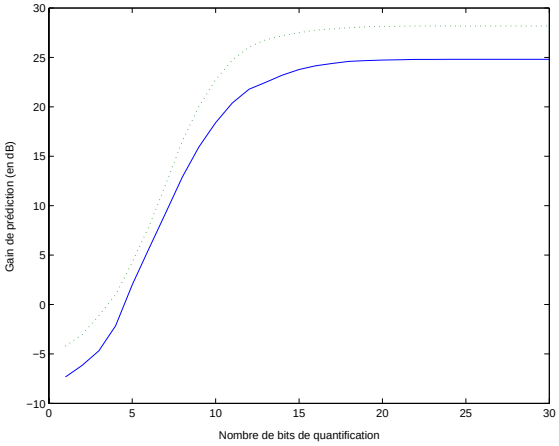


Fig. 3.21 - Gain de prédiction en fonction du nombre de bits de quantification pour le biai de la seconde couche (voie gauche : —, voie droite : - - -)

Les coefficients d'un réseau de neurone sont donc beaucoup plus sensibles à la quantification que les coefficients d'un prédicteur linéaire : pour quantifier un paramètre du réseau, il faut presque deux fois plus de bits que pour quantifier un coefficient d'un prédicteur linéaire stéréo. De plus le réseau a deux fois plus de paramètres qu'un prédicteur linéaire : 132 au lieu de 61. Pour des trames de 1024 échantillons le débit lié aux coefficients serait alors de 1.29 bits/échantillons contre 0.36 bits/échantillons (si les coefficients du prédicteur linéaire sont codés sur 12 bits et ceux du prédicteur non linéaire codés sur 20 bits). Le gain apporté par le réseau MLP, même si il est relativement

important, sera donc perdu au final à cause de l'important débit lié aux coefficients a transmettre. On envisagera donc dans la partie suivante une configuration backward.

3.3 Prédiction non linéaire stéréo avec un réseau MLP : approche backward

Nous avons vu dans la partie précédente qu'une prédiction non linéaire à base de réseau de neurones MLP donnait de bien meilleurs gains de prédiction qu'une prédiction linéaire. Malheuresement, les paramètres étant très sensibles à la quantification, il est nécessaire de les coder sur un nombre important de bits, ce qui rend un réseau de neurone MLP inutilisable dans un codeur forward. C'est pourquoi, nous allons étudier dans cette partie une approche backward du prédicteur.

3.3.1 Les réseaux utilisés, algorithme d'apprentissage

On utilise les même réseaux que pour l'approche forward :

- on prend $P_1 = 20$ et $P_1 = 10$
- on choisit des réseaux à deux neurones cachés. On a ici aucune limite sur le nombre de paramètres du prédicteur car on a pas besoin de les transmettre mais ce choix est fait pour limiter la complexité des calculs.

L'algorithme d'apprentissage utilisé est Levenberg-Marquardt car il est le plus performant comme on l'a vu dans 3.2.2.

3.3.2 Le surapprentissage

Dans [26], le problème du surapprentissage ("overtraining") est mis en évidence. Dans une configuration "backward", le réseau est entraîné sur une trame et testé sur la suivante. Si le réseau est parfaitement adapté à la trame utilisée pour l'apprentissage, il se peut qu'il ne le soit plus à la trame de test : on dit alors qu'il y a surapprentissage. Le réseau ne doit donc pas être spécifique aux données utilisées pour l'apprentissage, il doit être capable de généraliser. Pour éviter ce problème de surapprentissage, il faut que le nombre d'époques utilisées pour l'apprentissage ne soit pas trop élevé. Il ne doit pas non plus être trop petit car le réseau n'aurait alors plus aucun pouvoir de prédiction.

La figure suivante montre le gain de prédiction d'une trame du fichier clapton en fonction du nombre d'époques pour différentes initialisations aléatoires (zoom).

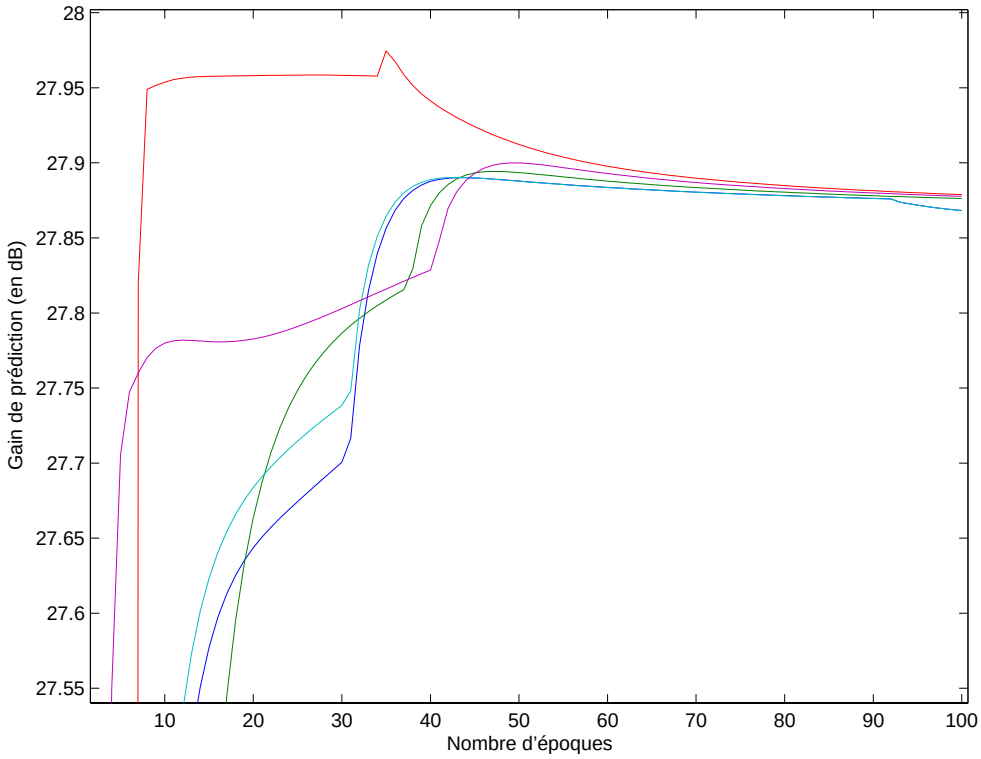


Fig. 3.22 - Gain de prédiction en fonction du nombre d'époques pour une trame du fichier clapton et pour différentes initialisations aléatoires (zoom)

On voit bien que le gain de prédiction diminue à partir d'un certain nombre d'époques. Le choix du nombre d'époques sera donc primordial.

3.3.3 Mise en oeuvre

On prend une longueur de trame égale à 1024 échantillons. On effectue une analyse en sous-trames de 256 échantillons. Les tests sont effectués sur des extraits (25 trames) des fichiers clapton et vega.

On utilise la technique définie dans 3.2.2 (multi-start) : pour chaque trame, on effectue 4 initialisations, 1 correspondant au réseau déterminé pour la trame précédente. On choisit celle qui donne le meilleure gain de prédiction, le gain étant celui de la trame utilisée pour l'apprentissage. Les 2 figures suivantes montrent la moyenne des gains de prédiction obtenus en fonction du nombre d'époques.

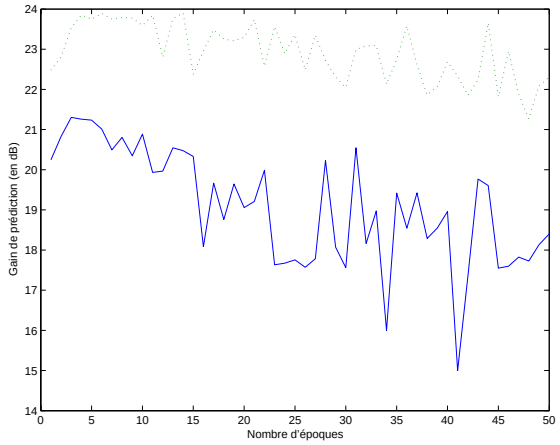


Fig. 3.23 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton
(voie gauche : —, voie droite - - -)

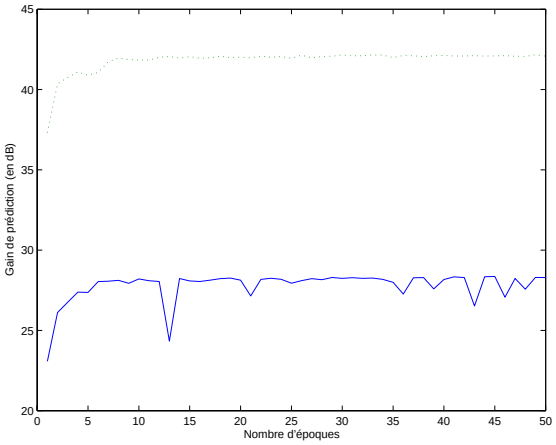


Fig. 3.24 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega
(voie gauche : —, voie droite - - -)

On remarque que le surapprentissage peut être très marqué (cas du fichier clapton).

- Fichier Clapton : le maximum est atteint pour 3 époques sur la voie gauche (21.3 dB) et 6 époques sur la voie droite (23.9 dB).
- Fichier Vega : le maximum est atteint pour 45 époques sur la voie gauche (28.4 dB) et 49 époques sur la voie droite (42.1 dB).

3.3.4 Techniques pour limiter le surapprentissage

On trouve dans la littérature deux principales techniques pour limiter le surapprentissage : la régularisation bayesienne et la validation. La troisieme technique présentée est une adaptation du multi-start à l'approche backward.

La régularisation bayesienne

Dans [47], il a été montré que la valeur des poids était plus importante que leur nombre afin de limiter le surapprentissage. On montre, que si un grand réseau est utilisé et que l'algorithme d'apprentissage trouve une erreur quadratique moyenne faible avec des poids de valeurs absolues faibles, alors les performances en généralisation dépendent de la taille des poids plutôt que de leur nombre. Une idée a donc été de remplacer la fonction d'erreur de l'algorithme d'apprentissage par l'erreur quadratique régularisée :

$$msereg = \alpha.mse + \beta \sum_{j=1}^{n} w_j^2$$

Une minimisation de cette fonction d'erreur permet d'obtenir une valeur absolue des poids du réseaux relativement faible et donc, pour la raison énoncée précédemment, cela permet de limiter le surapprentissage. Le problème se trouve dans l'estimation des paramètres α et β. Il est possible de combiner l'estimation de α et β avec Levenberg-Marquardt grâce à une analyse bayesienne : c'est l'algorithme de Levenberg-Marquardt avec régularisation bayesienne, étudié en détail dans [46].

Mise en oeuvre On effectue les même tests que dans la partie précédente mais avec l'algorithme de régularisation bayesienne. Les 2 figures suivantes montrent la moyenne des gains de prédiction obtenus en fonction du nombre d'époques (avec les résultats obtenus dans la partie précédente pour comparaison).

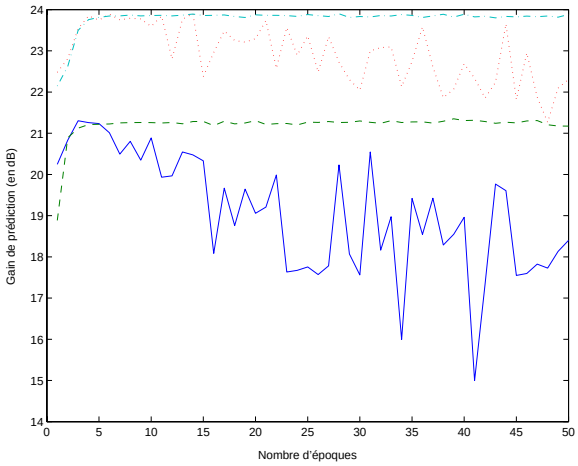


Fig. 3.25 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton
(voie gauche : —, voie droite - - -)

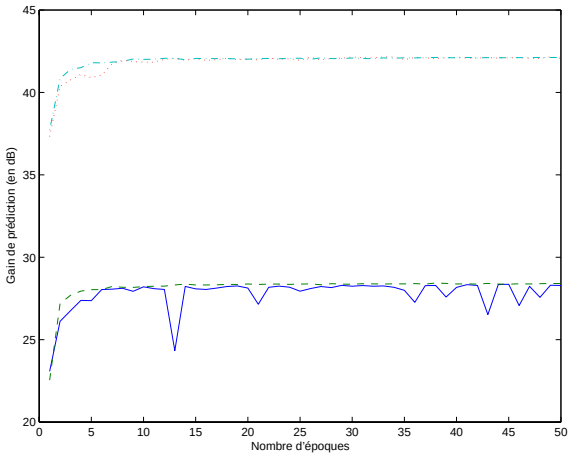


Fig. 3.26 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega
(voie gauche : —, voie droite - - -)

Cette technique limite donc bien le surrapprentissage. On obtient les gains de prédiction suivants :

- Fichier Clapton : le plafond est autour de 21.3dB pour la voie gauche et 23.8dB pour la voie droite
- Fichier Vega : le plafond est autour de 28.4dB pour la voie gauche et 42.1dB pour la voie droite

La validation

Cette technique consiste à calculer l'erreur d'une trame supplémentaire appelée trame de validation pendant l'apprentissage. Cette erreur devrait diminuer avec les époques. Mais dès qu'elle commence à augmenter, on arrête l'apprentissage, car cela veut dire qu'il y a surapprentissage. On pourra choisir comme trame de validation la trame qui précède la trame utilisée pour l'apprentissage ou encore la sous-trame qui précède la sous-trame de test. Le tableau suivant montre les gains

de prédiction obtenus. Notons que l'on a limité le nombre d'époques à 100 lorsqu'il n'y avait pas validation.

	Trame de validation	
	Sous-trame qui précède la sous-trame de test	Trame qui précede la trame d'apprentissage
Clapton	G : 18.8dB D : 22.5dB	G : 18.9dB D : 23.1dB
Vega	G : 28.3dB D : 42.1dB	G : 28.1dB D : 41.7dB

Tab. 3.2 - Gains de prédiction - Validation

L'avantage de cette technique est qu'il n'y a pas à fixer au préalable un nombre d'époques commun à toutes les trames. Mais, malheuresement, les gains obtenus sont inférieurs à ceux obtenus avec l'algorithme de Levenberg-Marquardt avec régularisation bayesienne.

Amélioration du multi-start

Dans la partie précédente, le choix de la meilleure initialisation du multi-start se fait à partir du gain de la trame utilisée pour l'apprentissage. On pourra envisager une technique basée sur le même principe mais où le choix de la meilleure initialisation se fera à partir du gain de prédiction de la trame de test. L'inconvénient de cette technique est que le choix de l'initilisation devra être transmis au décodeur : 2 bits pour chaque canal.

Les 2 figures suivantes montrent la moyenne des gains de prédiction obtenues en fonction du nombre d'époques (avec les résultats obtenus dans la partie précédente pour comparaison).

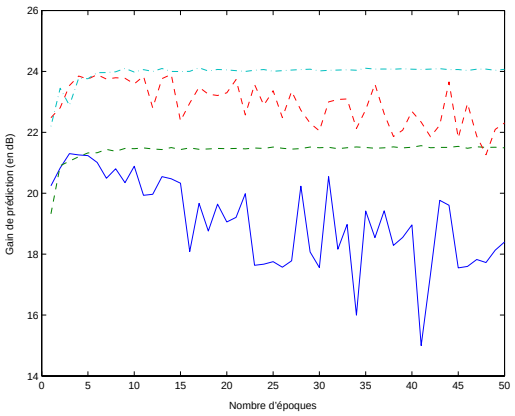


Fig. 3.27 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton
(voie gauche : —, voie droite - - -)

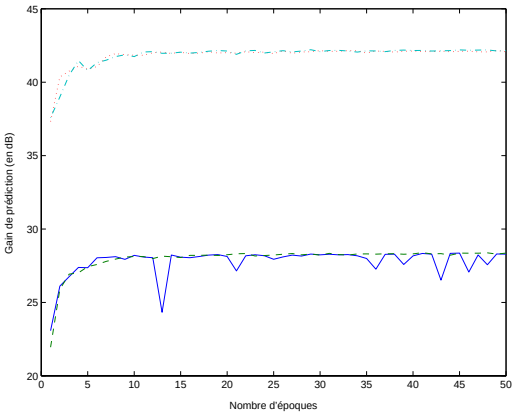


Fig. 3.28 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega
(voie gauche : —, voie droite - - -)

Le surapprentissage est dans ce cas beaucoup moins prononcé et on obtient même de meilleurs résultats qu'avec l'algorithme de régularisation bayesienne.

– Fichier Clapton : le plafond est autour de 21.5dB pour la voie gauche et 24.1dB pour la voie droite
– Fichier Vega : le plafond est autour de 28.4dB pour la voie gauche et 42.1dB pour la voie droite

Cette technique est donc idéale et c'est celle-ci que l'on retiendra par la suite.

3.3.5 Influence du nombre de neurones

Dans l'approche backward, il n'y a aucune limite sur le nombre de paramètres du prédicteur car on n'a pas besoin de les transmettre. Le nombre de neurones du réseau n'est donc limité que par la complexité des calculs.

Les figures suivantes montre la moyenne des gains de prédiction obtenus en fonction du nombre d'époques pour des réseaux à 1 , 2 et 4 neurones cachés.

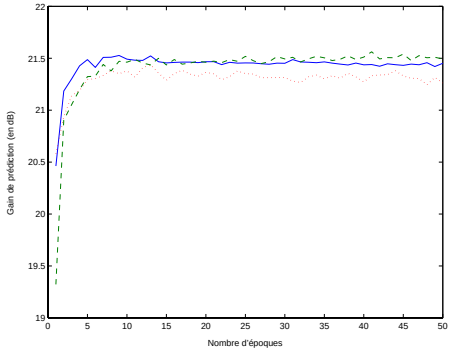


Fig. 3.29 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton - voie gauche
(1 neurone : —, 2 : - - -, 4 : · · ·)

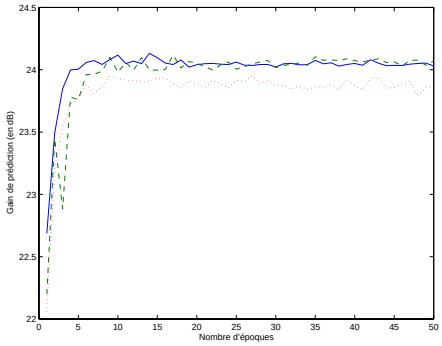


Fig. 3.30 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton - voie droite
(1 neurone : —, 2 : - - -, 4 : · · ·)

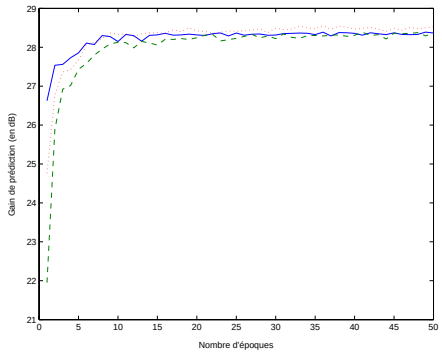


Fig. 3.31 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega - voie gauche
(1 neurone : —, 2 : - - -, 4 : · · ·)

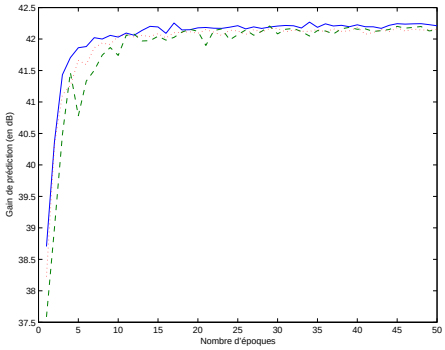


Fig. 3.32 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega - voie droite
(1 neurone : —, 2 : - - -, 4 : · · ·)

Dans l'approche forward, plus le réseau a de neurones, plus il est performant. Mais nous remarquons ici que cela ne s'applique pas au cas backward : plus le réseau a de neurones, plus il sera adapté à la trame utilisée pour l'apprentissage et moins il sera capable de généraliser. En effet, on remarque qu'un réseau avec seulement un neurone caché donne presque toujours un meilleur gain de prédiction (sauf pour la voie gauche de vega).

3.3.6 Comparaison avec un apprentissage simple

On compare la technique retenue précédemment avec un apprentissage simple qui ne réinitialise pas le réseau pour chaque sous-trame. La seule initialisation du réseau à réaliser est alors pour la première trame : pour ne pas utiliser de générateur aléatoire, on initialisera les poids de la première couche à 0 et ceux de la seconde couche à 1 (biais à 0). Les figures suivantes montre la moyenne des gains de prédiction obtenus en fonction du nombre d'époques pour des réseaux à 1 , 2 et 4 neurones cachés.

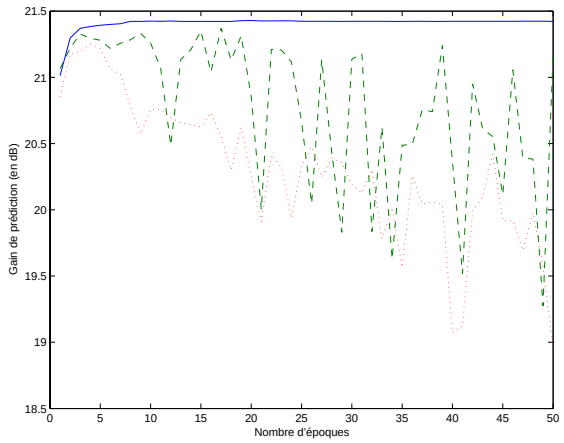


Fig. 3.33 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton - voie gauche (1 neurone : —, 2 : - - -, 4 : · · ·)

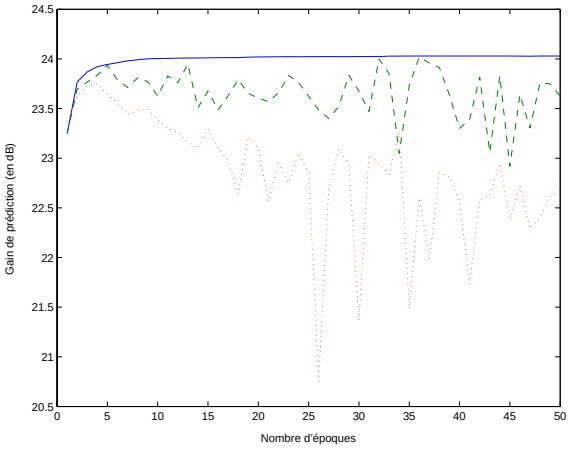


Fig. 3.34 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier clapton - voie droite (1 neurone : —, 2 : - - -, 4 : · · ·)

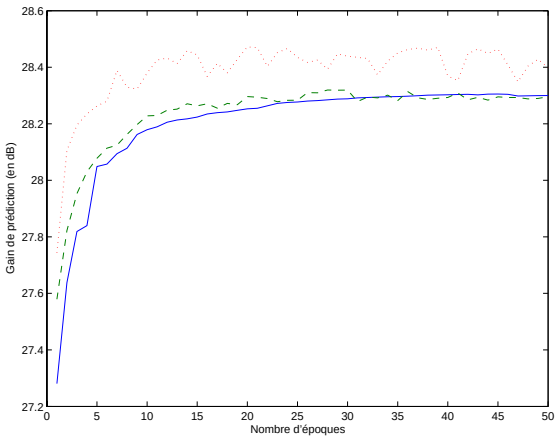


Fig. 3.35 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega - voie gauche (1 neurone : —, 2 : - - -, 4 : · · ·)

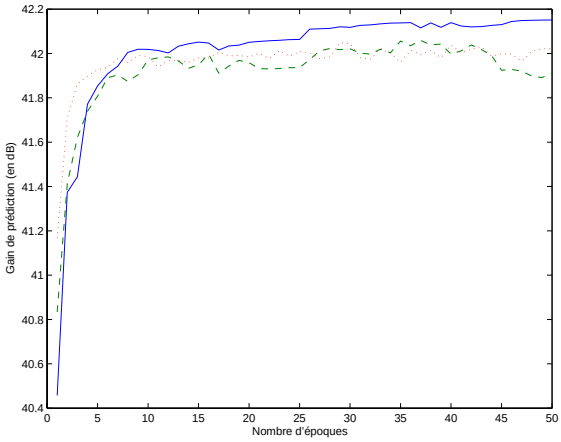


Fig. 3.36 - Gain de prédiction en fonction du nombre d'époques pour un extrait du fichier vega - voie droite (1 neurone : —, 2 : - - -, 4 : · · ·)

On remarque dans un premier temps que, comme dans 3.3.5, un réseau à 1 neurone donne de meilleurs gains de prédiction sauf pour la voie gauche de vega. Le cas du fichier clapton montre bien que plus le réseau à de neurones, plus le surapprentissage est marqué.

On remarque ensuite que la technique retenue précédemment (multi-start modifié) permet de gagner entre 0.1 et 0.2 dB pour un réseau à un neurone caché.

3.3.7 Comparaison avec la prédiction linéaire stéréo backward

On teste sur les 2 extraits utilisés précédemment l'apprentissage retenu dans la partie précédente. Les réseaux ont donc un seul neurone et le nombre d'époques sera fixé à 10.

Les figures suivantes montrent le gain par rapport à une prédiction linéaire stéréo pour les 100 sous-trames de chaque voie.

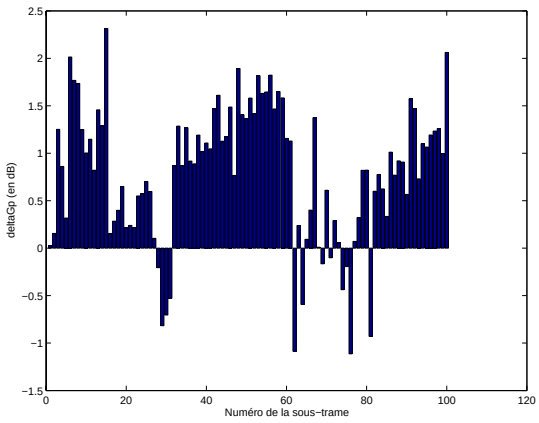


Fig. 3.37 - Voie gauche de clapton

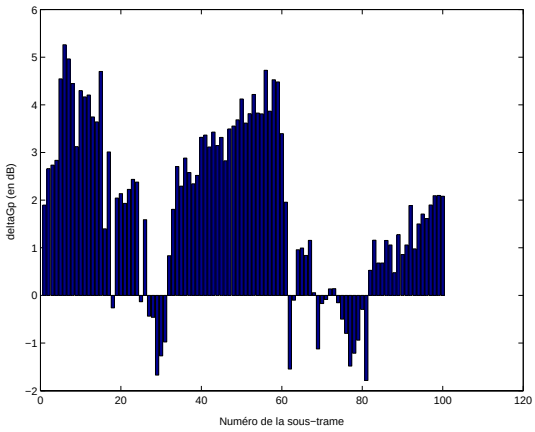


Fig. 3.38 - Voie droite de clapton

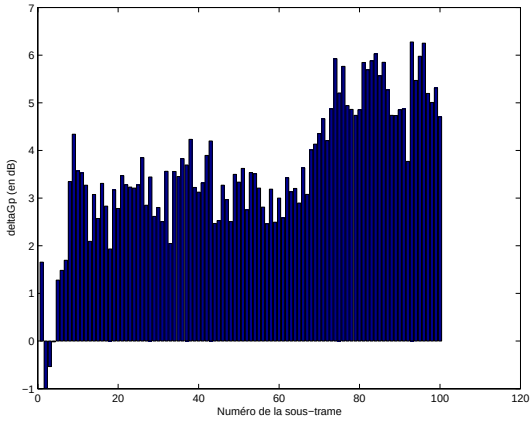


Fig. 3.39 - Voie gauche de vega

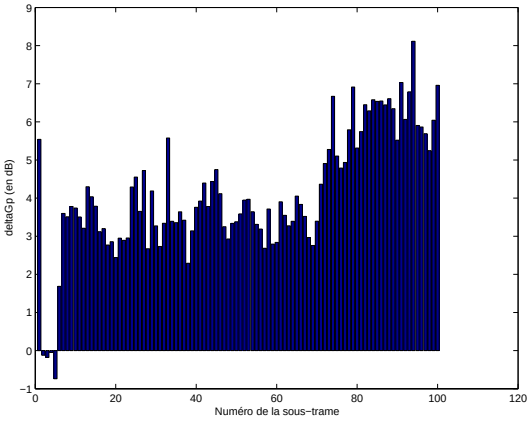


Fig. 3.40 - Voie droite de vega

La prédiction avec un réseau MLP apporte donc un gain significatif par rapport à une prédiction linéaire. On remarque aussi que pour un petit nombre de sous-trames, la prédiction linéaire donne de meilleurs gains de prédiction. Nous envisagerons donc la technique qui consiste à choisir, pour chaque sous-trame, la prédiction qui donne le meilleur gain de prédiction. Pour que le décodeur sache quelle prédiction a été choisie au niveau du codeur, il faudra transmettre un bit par sous-trame et par voie.

3.4 Conclusions

Le but de ce chapitre était de voir si une prédiction basée sur un réseau de neurones donnait de meilleurs gains de prédiction qu'une simple prédiction linéaire. Nous avons d'abord étudié une approche forward du prédicteur stéréo puis une approche backward. Nous sommes parvenu aux conclusions suivantes :

- Un prédicteur stéréo basé sur un réseau de neurones donne de meilleurs gains de prédiction qu'une prédiction stéréo linéaire en forward. Mais ce type de prédiction coute trop cher en terme de débit à cause du nombre important de coefficients à transmettre ainsi que du nombre important de bits nécessaires à la quantification.
- En backward, un prédicteur stéréo basé sur un réseau de neurones améliore de manière significative les gains de prédiction par rapport à une prédiction stéréo linéaire. On a vu que le multi-start amélioré est la technique qui donnait les meilleurs résultats. Cependant, on a vu aussi qu'un apprentissage qui n'utilisait pas de multi-start perdait seulement entre 0.1 et 0.2 dB en gain de prédiction. Comme le multi-start nécessite 4 fois plus de calculs, ce gain n'est pas forcément justifié.

Chapitre 4

Codec à prédiction linéaire et non linéaire stéréo backward

Aprés avoir étudié un certain nombre de méthodes de prédiction non linéaire, nous avons retenu celle qui donnait les meilleurs résultats : une prédiction stéréo avec un réseau de neurones MLP en backward. De plus, nous avons vu qu'un réseau avec seulement un seul neurone donnait de meilleurs résultats et qu'il n'était pas nécessaire d'envisager un apprentissage multi-start étant donné le faible gain qu'il apporte et la grande complexité des calculs qu'il nécessite. On a vu aussi que pour obtenir un gain de prédiction optimal, il faut réaliser deux analyses pour chaque sous-trame : une linéaire et une avec un réseau de neurones et choisir celle qui donne le meilleur gain de prédiction.

Nous allons décrire dans cette partie comment on intègre la technique retenue dans un codeur implémenté en C. Un codeur implémentant une prédiction linéaire stéréo avait déja été réalisé. Les codes ont donc été repris et complétés pour intégrer la prédiction non linéaire avec le réseau MLP. Une grande partie des descriptions des différentes parties du codeur sont donc des extraits de [1].

4.1 Les types de fichiers

Fichier d'entrée

Les fichiers d'entrée sont des fichiers binaires sans entête (contrairement aux fichiers .wav). Les voies gauches et droites sont entrelacées.

Fichier de sortie

Le fichier compressé est composé de :

– une entête : comprenant le nombre total de trames ainsi que le nombre d'échantillons de la dernière trame

– pour chaque trame : les résidus encodés par le code de Rice ainsi que 2 bits transmis au décodeur pour qu'il sache quelle prédiction le codeur a choisi pour chaque voie.

4.2 Schéma du codeur

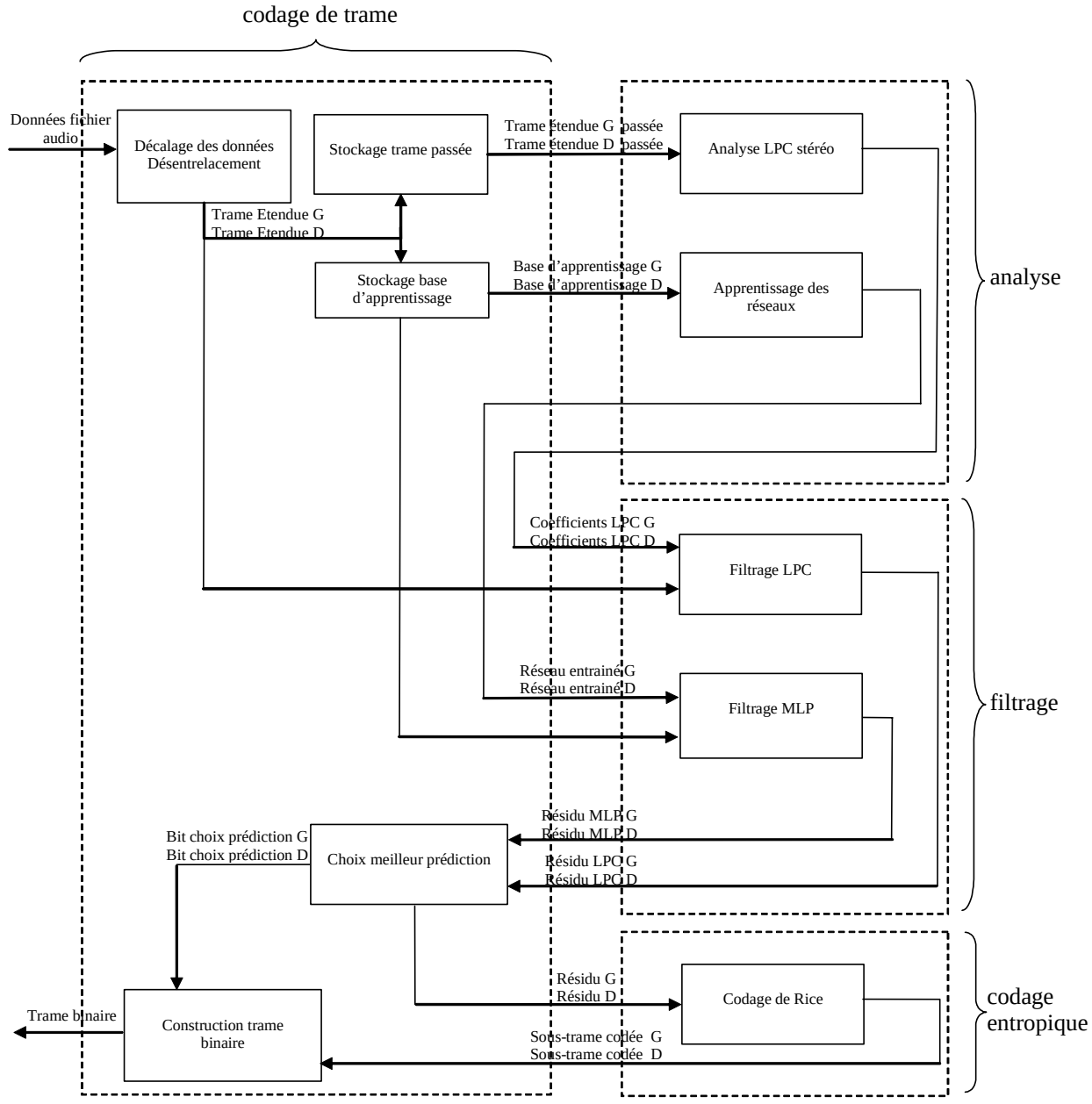


Fig. 4.1 - Schéma du codeur

4.2.1 Analyse par trame - Gestion des trames

Il s'agit ici de décrire les blocs de décalage des données - désentrelacement et de stockage de la trame passée. Etant donné que l'analyse se fait sur la trame qui précéde la trame à coder, il faut avoir à disposition deux trames consécutives. C'est pourquoi on travaillera avec une trame étendue comprenant deux trames consécutives ainsi q'un recouvrement paramétrable à gauche de la trame d'analyse afin de disposer d'un nombre de données suffisant pour le calcul des corrélations de la prédiction linéaire. Le système de roulement d'une trame à l'autre consiste simplement à décaler les données à gauche de manière à insérer la nouvelle trame à droite comme décrit sur la figure 4.2.

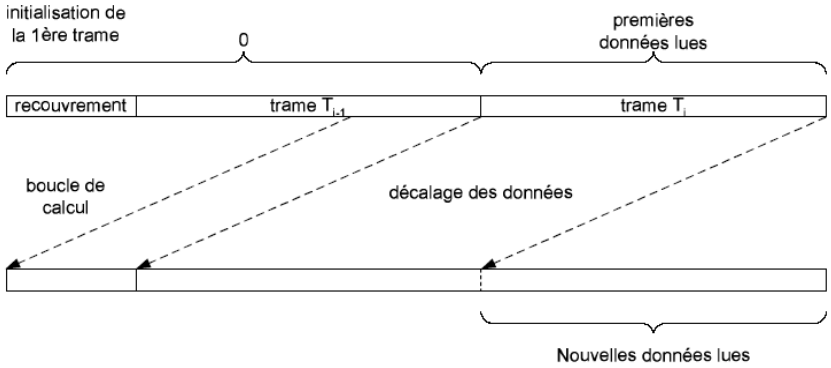


Fig. 4.2 - Gestion des trames

4.2.2 Analyse LPC stéréo backward

Fenêtre de pondération

Les coefficients de prédiction linéaire stéréo sont estimées sur la trame précédente et nécessitent de calculer les fonctions d'auto et intercorrélation, avec une fenétre de pondération adéquate. Plusieurs types de fenêtres ont été testés dans [2], et la fenêtre de Hamming assurait les meilleures performances. Cependant, cette fenétre, centrée sur la trame précédente donne plus d'importance à des échantillons passés d'une dizaine de millisecondes si la trame est de durée 20 ms. L'idée consiste alors à construire une fenêtre non symétrique, qui donne plus de poids à des échantillons plus récents (autour de 5 ms). Une méthode de fenêtre hybride est mise en oeuvre dans [2]. La fenêtre retenue pour le codeur backward, illustrée par la figure 6.3 se divise en 2 parties :

- Une partie gauche identique à une demi fenêtre de Hamming (95 % de la fenêtre).
- Une partie droite en quart de période de cosinus, dont la retombée est rapide (5% de la fenêtre).

Analyse LPC stéréo

Les sous blocs des matrices R_1 et R_2 sont construits facilement du fait de leur propriétés (matrice dites Toeplitz) à partir des vecteurs des autocorrélations et intercorrélations. L'inversion des matrices R_1 et R_2 est une implémentation de la librairie de fonctions LAPACK. Il s'agit d'une librairie

de fonctions initialement développées dans des universités en langage Fortran 77, permettant de résoudre efficacement des problèmes d'algèbre linéaire. Il existe aujourd'hui une version transposée en C, dont l'utilisation nécessite de prendre quelques précautions, notamment lorsqu'on doit passer des matrices. En effet, en langage C, une matrice doit être passée comme un seul vecteur, les lignes étant concaténées. En Fortran 77, une matrice est passée également par un vecteur, mais dont les colonnes sont mises bout à bout. Les matrices R_1 et R_2 que l'on doit décomposer par Cholesky et inverser doivent donc être passées en colonnes aux fonctions de la librairie. Les matrices R_1 et R_2 sont conditionnées en ajoutant un bruit fictif sur les diagonales. Cette opération est indispensable, sous peine de ne pas pouvoir inverser les matrices (risques de valeurs propres nulles).

Analyse en sous-trames

Comme il n'y a pas de coefficients à transmettre, l'analyse LPC stéréo backward peut être renouvelée aussi souvent que souhaité. On peut donc répéter le calcul des coefficients de prédiction sur un certain nombre de sous-trames, comme cela est illustré par la figure 4.3. Le modèle LPC est calculé sur la zone pondérée par la fenêtre et appliqué sur la sous-trame grisée. Cette fréquence d'analyse permet d'adapter le modèle plus souvent sans affecter le débit. Cependant, la charge de calcul est plus importante.

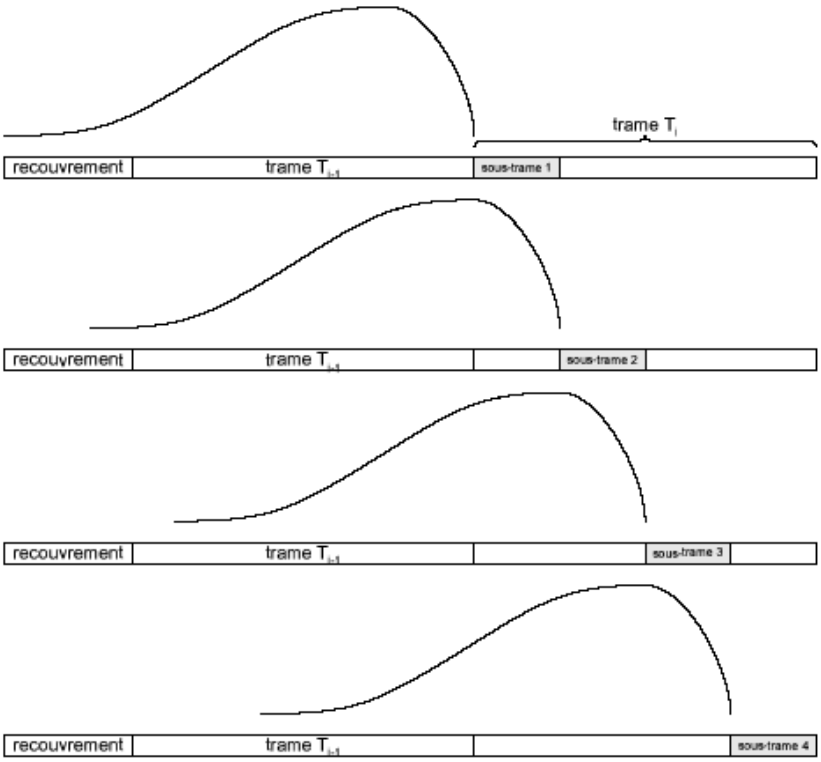


Fig. 4.3 - Analyse backward sur des sous-trames

4.2.3 Apprentissage des réseaux MLP

Pour implémenter un réseau de neurones MLP, on a utilisé une librairie écrite en C++. Cette librairie est le code source d'un livre sur des méthodes avancées d'apprentissage de réseaux de neurones []. Elle a été choisie car c'était le seul code trouvé sur internet qui était gratuit et qui implémentait l'algorithme de Levenberg-Marquardt. On utilise 2 classes de cette librairie :

- La classe **TrainingSet** qui implémente une base d'apprentissage.
- La classe **LayerNet** qui implémente un réseau MLP ainsi que l'algorithme de Levenberg-Marquardt.

La classe TrainingSet

Elle est composée des attributs suivants :

- Un certain nombre d'informations comme la taille du patron d'entrée, la taille du patron de référence ainsi que le nombre de couples d'apprentissage.
- Les données de la base d'apprentissage. Ses données sont rangées dans un tableau selon le schéma suivant :

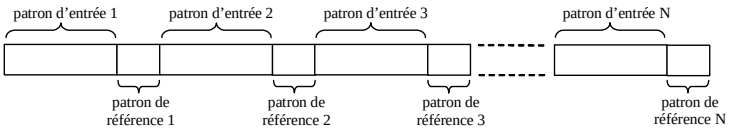


Fig. 4.4 - Structure d'une base d'apprentissage

La classe LayerNet

Les attributs de la classe LayerNet sont toutes les informations nécessaires à la création du réseau MLP : le mode (réel ou complexe), le nombre de couches, le nombre de neurones de chaque couches, la fonction de transfert du neurone de sortie. Le constructeur initialise ces paramètres, alloue la mémoire nécessaire et initialise les poids du réseau (biais à 0, poids de la première couche à 0 et poids de la seconde couche à 1).

La classe comprend aussi un certain nombre de méthodes mais nous n'en utiliserons que 3 :

- la méthode **lev_marq**, qui entraine le réseau avec l'algorithme de Levenberg-Marquardt. Elle a comme entrées une base d'apprentissage du type TrainingSet ainsi que le nombre d'époques.
- la méthode **trial**, qui calcule la sortie du réseau pour une entrée donnée.

Gestion de la base d'apprentissage

On utilisera un buffer pour stocker la base d'apprentissage. Ce buffer comprend la base d'apprentissage elle-même mais aussi la base de test correspondante. En effet, la base de test d'une sous-trame donnée fait partie de la base d'apprentissage de la sous-trame suivante. Le passage d'une

sous-trame à une autre consiste alors à décaler les données du buffer à gauche de manière à insérer la nouvelle base de test à droite.

L'apprentissage est alors réalisé avec la première partie du buffer (base d'apprentissage), la deuxième partie (base de test) sera utilisé pour le filtrage.

4.2.4 Filtrage et choix de la prédiction

Filtrage LPC

On réalise ici le filtrage LPC. Les calculs étant réalisés sur des nombres flottants de précision 32 bits, il convient de porter une attention particulière aux arrondis. En effet, le résidu de prédiction doit être entier afin d'être compressé par un codeur entropique. Si cet arrondi n'est pas correctement réalisé, il sera impossible de retrouver au bit prés le fichier original au décodage.

Filtrage avec le réseau MLP

Il s'agit simplement de présenter au réseau les patrons d'entrées de la base de test et de calculer la sortie grâce à la méthode trial. Comme pour le filtrage LPC, on arrondit à l'entier le plus proche la valeur prédite.

Choix de la prédiction

Pour chaque sous-trame, on calcule un résidu avec une prédiction LPC, un résidu avec le réseau MLP ainsi que l'énergie de chaque résidu. On choisit la prédiction qui donne le résidu d'énergie minimale. Un bit pour chaque voie est ajouté à la trame codée pour que le décodeur sache quelle prédiction réaliser.

4.2.5 Codage entropique

Caractéristiques du codeur entropique.

Il s'agit d'un code de Rice tel qu'il est décrit dans 1.7.2. Nous avons vu que dans ce type de codeur, le paramètre de Golomb est adaptatif. Cependant, la dynamique des résidus à coder peut varier beaucoup sur la longueur d'une trame, c'est pourquoi il est conseillé de remettre à jour ce paramètre à une fréquence supérieure à la fréquence de trame. Ainsi, on pourra calculer le paramètre optimal de Golomb sur chaque sous-trame.

Paramètre de Golomb et encodage

Une série de 32 paramètres de Golomb adaptés à la dynamique maximale des résidus est stockée dans un vecteur. L'algorithme va alors identifier la valeur maximale du résidu à coder sur la sous-trame et en déduire un paramètre de Golomb. Chaque sous-trame de chaque canal est alors encodée en une trame binaire de longueur variable, avec en entête le code sur 5 bits du paramétre de Golomb.

4.2.6 Construction de la trame binaire

Les résidus de chaque sous-trame sont assemblés dans une même trame binaire dont la structure est donnée par la figure 4.5.

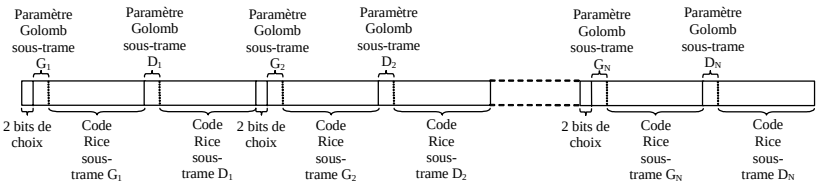


Fig. 4.5 - Construction de la trame binaire

4.3 Schéma du décodeur

4.3.1 Gestion du buffer des données compressées

Le décodeur va ainsi d'abord lire l'entête du fichier compressé dont la longueur est connue : elle contient le nombre de trames à décoder et la longueur de la dernière trame. Ensuite, la solution retenue consiste à envoyer à la fonction de décodage de trame un long buffer dont on est sûr qu'il contient suffisamment de bits pour décoder une trame stéréo. La composition de la trame binaire et le nombre d'échantillons à décoder sont connus. Un compteur détermine quelle trame est décodée, afin de prévoir le cas particulier de la dernière trame. Une fois que la trame de signal stéréo est reconstruite, le programme décale les données dans le buffer des bits compressés et réclame au programme principal de compléter ce buffer avec des nouvelles données. Ce processus s'apparente à un buffer circulaire. Un autre buffer permet de conserver en mémoire les données audio de la trame précédente.

4.3.2 Analyse LPC stéréo au décodage

Une analyse LPC stéréo doit être réalisée sur la trame précédemment décodée afin d'en déduire les paramètres LPC qui vont permettre de décoder la sous-trame de résidus présente. Lorsqu'une trame a été décodée, les données sont décalées vers la gauche. Ensuite, des analyses LPC stéréo sont réalisées et appliquées aux sous-trames : toutes ces opérations sont identiques aux opérations réalisées dans la partie analyse LPC du codeur.

4.3.3 Apprentissage des réseaux au décodage

L'apprentissage est gérer de la même facon qu'au codage.

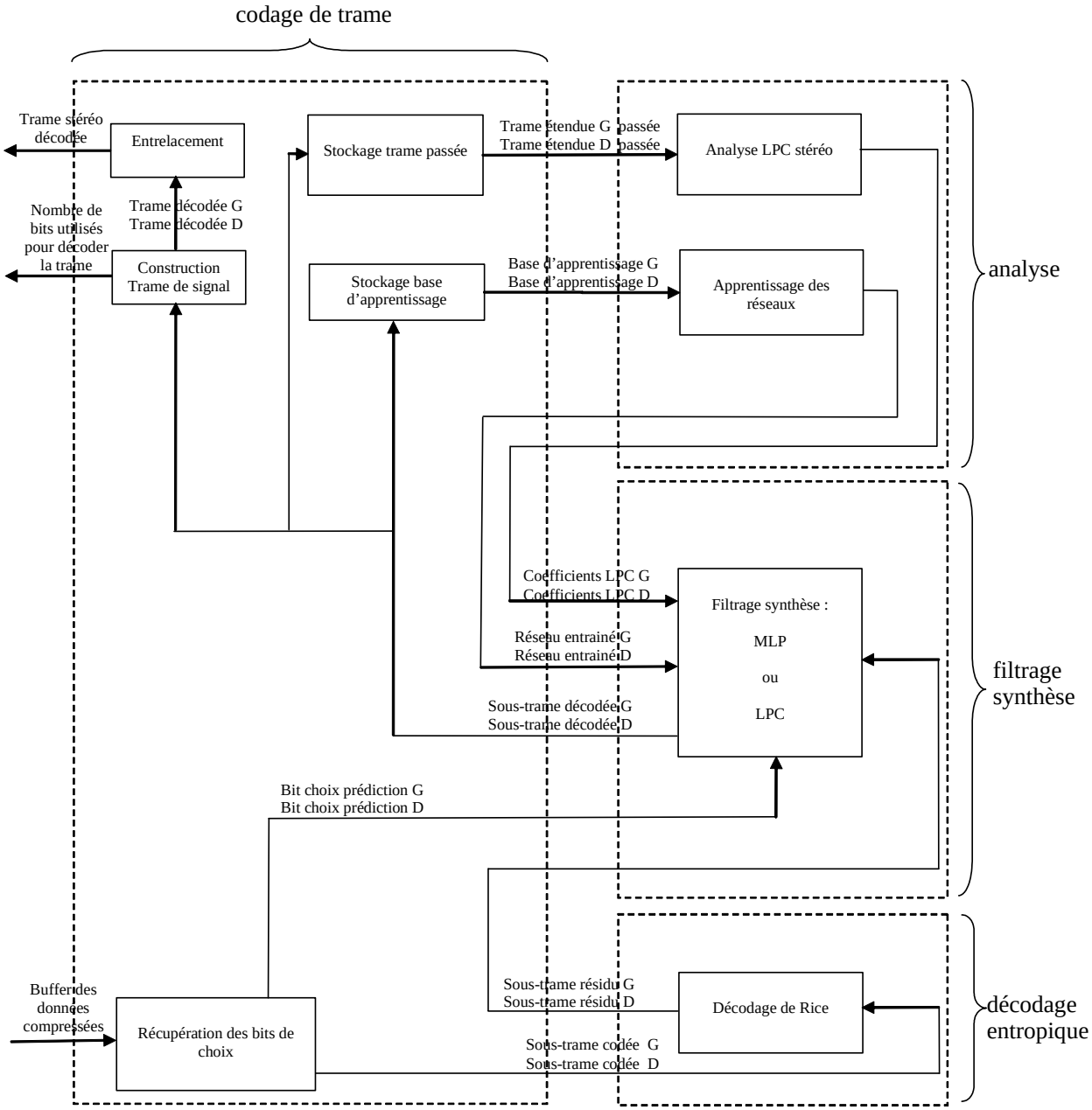


Fig. 4.6 - Schéma du décodeur

4.3.4 Décodage de Rice

On récupère d'abord les 2 bits de choix de la prédiction ainsi que les 2 bits pour le filtrage NLMS. Le pointeur sur le buffer de données binaire est déplacé du nombre de bits qui sont utilisés pour le décodage. D'après la configuration de la trame compressée, les 5 premiers bits correspondent au

paramètre de Golomb de la première sous-trame gauche de N_{ST} échantillons. Une fois le paramètre de Golomb connu, le module de décodage de Rice peut décoder échantillon par échantillon jusquà ce qu'il ait atteint N_{ST} échantillons. L'opération est alors répétée pour la première sous-trame droite, et ainsi de suite jusqu'à la dernière sous-trame droite.

4.3.5 Filtrage de synthèse et entrelacement

Les signaux obtenus sont filtrés avec le filtre choisi au niveau du codeur. Lorsque toutes les sous-trames ont été successivement décodées, il est alors possible d'entrelacer les données décompressées.

4.4 Evaluation des performances

4.4.1 Influence de l'ordre

On cherche dans un premier temps à voir l'influence que peut avoir l'ordre des auto- et interprédicteurs sur le ratio de compression. Pour cela, on fixe le nombre d'époques des 2 réseaux à 3. Les tests sont effectués sur un extrait du fichier clapton. Les résultats sont présentés dans le tableau 4.1.

Ordres P_1/P_2	Ratio	Ordres P_1/P_2	Ratio
20/10	1.6182	40/30	1.6333
20/20	1.6227	40/40	1.6332
30/10	1.6238	50/10	1.6292
30/20	1.6281	50/20	1.6325
30/30	1.6293	50/50	1.6345
40/10	1.6268	60/60	1.6342
40/20	1.6320		

Tab. 4.1 - Influence de l'ordre des auto- et interprédicteurs sur le ratio de compression

Le maximum est obtenu pour la combinaison $P_1 = 50/P_2 = 50$. Mais dans ce cas, les calculs nécessaires à l'apprentissage deviennent trés complexe. On se limitera donc à la combinaison $P_1 = 40/P_2 = 20$ qui semble être un bon compromis entre la performance et la complexité des calculs.

Ordres retenus : $P_1 = 40/P_2 = 20$

4.4.2 Influence du nombre d'époques

On étudie dans un second temps l'influence du nombre d'époques de l'apprentissage des réseaux sur le ratio de compression. Le fichier utilisé est la base de test MPEG. Les résultats sont présentés dans le tableau 4.2.

Nombre d'époques réseau G / réseau D	Ratio
1	2.5125
2	2.5510
3	2.5671
4	2.5695
5	2.5702
6	2.5721
7	2.5732
8	2.5735
9	2.5741
10	2.5744
15	2.5744

Tab. 4.2 - Influence du nombre d'époques sur le ratio de compression

On atteint un plafond pour un nombre d'époques égale à 10. On réalisera donc l'apprentissage des réseaux avec un nombre d'époques égale à 10.

Nombre d'époques retenu : 10

4.4.3 Confrontation des performances

On compare le codec développé au cours du stage avec :

- le même codec mais sans la prédiction non linéaire. C'est le codec qui utilise une prédiction stéréo linéaire backward, et qui a été développé pendant le stage de Jean-Luc Garcia [1]
- les codecs LPAC et Monkeys, qui sont les codecs de l'état de l'art les plus performants

Base de fichiers MPEG

Cette base est constituée de 12 fichiers déja utilisés dans les parties précédentes. Ces fichiers ont pour particularité de présenter une dynamique relativement faible.

Algorithme	LPAC	Monkeys	Linéaire	MLP
Ratio	2.503	2.573	2.381	2.574

Tab. 4.3 - Comparaison des résultats sur la base de test MPEG

Le gain est significatif : alors que le codec qui utilise une prédiction linéaire se situe en dessous des performances de LPAC et Monkeys, notre codec obtient le meilleur ratio de compression.

Autre base de fichiers

Ces tests sont réalisés sur des fichiers de musique de styles différents : pop, jazz, classique, rock, électronique...

On voit que les ratios obtenus avec notre codec sont tous supérieurs aux ratios obtenus avec le codec qui utilise une prédiction linéaire. De plus, les ratios obtenus sont tous supérieurs à ceux de LPAC, et juste en dessous de ceux de Monkeys.

Algorithme	LPAC	Monkeys	Linéaire	MLP
Born in the USA	1.424	1.461	1.416	1.448
concerto	1.924	1.998	1.872	1.938
Cosmic girl	1.521	1.578	1.539	1.564
Luka	1.422	1.490	1.455	1.476
Polonaise	2.610	2.700	2.295	2.654
Training	1.582	1.621	1.568	1.604

Tab. 4.4 - Comparaison des résultats sur des fichiers de musique de styles différents

4.5 Amélioration du codeur

On a vu dans 2.3.2 qu'un filtre NLMS non linéaire ne donnait pas de meilleurs gains de prédiction qu'un filtre NLMS linéaire. Mais on a vu aussi que le filtre NLMS linéaire donnait des gains de prédiction compris entre 1 et 2 dB. Les résidus de la prédiction stéréo ne sont donc pas completement décorrélés et un filtrage NLMS permet d'enlever encore de la redondance dans le signal.

On peut donc améliorer notre codec en ajoutant un filtre NLMS sur les résidus de la prédiction stéréo. Pour optimiser au maximum la décorrélation des résidus, on va même réaliser un filtrage NLMS en cascade : deux filtres NLMS sur chaque voie.

Quelques tests réalisés sur la base de test MPEG ont montrés qu'un premier filtre d'ordre 200 (avec $\mu = 0.07$) et un second filtre d'ordre 10 (avec $\mu = 0.03$) donnaient les meilleurs résultats.

Confrontation des performances

On compare le codec amélioré avec le codec précédent et les 2 codecs de l'état de l'art.

Base de fichiers MPEG

Algorithme	LPAC	Monkeys	Linéaire	MLP	MLP + NLMS
Ratio	2.503	2.573	2.381	2.574	2.720

Tab. 4.5 - Comparaison des résultats sur la base de test MPEG

Autre base de fichiers

Algorithme	LPAC	Monkeys	Linéaire	MLP	MLP +NLMS
Born in the USA	1.424	1.461	1.416	1.448	1.461
concerto	1.924	1.998	1.872	1.938	2.000
Cosmic girl	1.521	1.578	1.539	1.564	1.578
Luka	1.422	1.490	1.455	1.476	1.491
Polonaise	2.610	2.700	2.295	2.654	2.701
Training	1.582	1.621	1.571	1.604	1.617

Tab. 4.6 - Comparaison des résultats sur des fichiers de musique de styles différents

Le codec amélioré donne donc les meilleurs performances, avec des ratios comparables à ceux de Monkeys.

Chapitre 5

Perspectives d'évolution

Dans le chapitre 4, nous avons etudié une prédiction non linéaire avec un reseau de neurones MLP. Mais le réseau MLP n'est pas le seul type de réseaux de neurones utilisé pour faire de la prédiction. On trouve dans la littérature deux autres types de réseaux : le réseau à fonction radiale de base et le réseau PRNN. Ces deux types de réseaux ont été étudiés dans le cadre du codage de la parole et donne de bons résultats. Une perspective d'évolution serait donc d'étudier ces réseaux dans le cadre du codage audio sans perte.

5.1 Le Réseau de Fonctions à Base Radiale (RBF network)

Ces réseaux ont été étudiés dans [36], [37], [39] et [40]. Ces articles montrent qu'une prédiction avec un réseau RBF donne de meilleurs gains de prédiction qu'une prédiction linéaire pour des signaux de parole.

5.1.1 Le réseau standard

Un réseau de Fonctions à Base Radiale a la même structure qu'un perceptron à une couche cachée (de taille K) et un neurone de sortie. Les cellules de sortie effectuent une combinaison linéaire de fonctions non linéaires, fournies par les neurones de la couche cachée. Ces fonctions produisent une réponse différente de zéro seulement lorsque l'entrée se situe dans une petite région bien localisée de l'espace des variables. Bien que plusieurs modèles de telles fonctions existent, le plus courant est de type gaussien :

Si on note le vecteur des entrées d'un neurone $\mathbf{i} = (i_i)_{i=1..n}$ et S sa sortie, on a :

$$S = \exp(-\frac{\sum_{i=1}^{n}(i_i - t_i)^2}{2\sigma^2})$$

$\mathbf{t} = (t_i)_{i=1..n}$ est appelée le centre du neurone et σ sa largeur. La relation d'entrée-sortie du réseau est donc :

$$S_{réseau} = \sum_{k=1}^{K} w_k \exp(-\frac{\|\mathbf{i} - \mathbf{t}_k\|^2}{2\sigma_k^2})$$

Plus le vecteur d'entrée est proche du centre d'une Gaussienne, plus la sortie du neurone de la première couche qui lui correspond est élevée. Le terme "Fonction à Base Radiale" vient du fait que la Gaussienne est symétrique radialement, c'est-à-dire que la valeur de sortie obtenue est la même pour toutes les entrées situées à une même distance du centre de la Gaussienne.

En théorie, le réseau de fonctions à base radiale est capable, tout comme le perceptron multicouches, d'effectuer une approximation arbitrairement proche de n'importe quelle transformation non linéaire. La principale différence entre les deux est la nature de la fonction d'activation des neurones de la couche cachée. La non-linéarité sigmoïdale utilisée dans le perceptron multicouches fournit une sortie différente de zéro pour une région infiniment grande de l'espace d'entrée, ce qui lui confère un certain pouvoir de généralisation dans des régions où peu de données d'apprentissage sont disponibles. Au contraire, la non-linéarité radiale utilisée ici, ne fournit une réponse différente de zéro que localement. Le nombre d'unités cachées nécessaire pour permettre au réseau de recouvrir l'ensemble de l'espace d'entrée peut dès lors s'avérer parfois trés élevé. Un autre inconvenient des réseaux RBF par rapport à des réseaux de type perceptron est qu'ils requièrent une base d'apprentissage plus large. Par contre, son principal avantage est que son apprentissage est plus simple et plus rapide.

5.1.2 Le réseau autorégressif (RBF-AR)

La relation entrée-sortie d'un réseau autorégressif est :

$$S_{réseau} = \sum_{d=0}^{D} w_{0,d} i_d + \sum_{k=1}^{K} \sum_{d=0}^{D} w_{k,d} i_d \exp(-\frac{\|\mathbf{i} - \mathbf{t}_k\|^2}{2\sigma_k^2})$$

5.1.3 L'apprentissage du réseau

L'apprentissage du réseau se fait en trois étapes. Les deux premières sont non-supervisées et la dernière est supervisée.

La procédure est la suivante :

1. Calcul des centres $\mathbf{t}_k$ par un algorithme de quantification vectorielle (par ex. la k-moyenne).
2. Calcul des largeurs σ_k en calculant l'écart type associé à l'ensemble des vecteurs d'entrée correspondant à chaque centre t_k et en identifiant un facteur de lissage r optimal :

$$\sigma_k = r.std_j$$

3. Calcul des poids w par la résolution d'un système d'équation linéaire en minimisant un critère d'erreur E (par. ex. l'erreur quadratique moyenne).

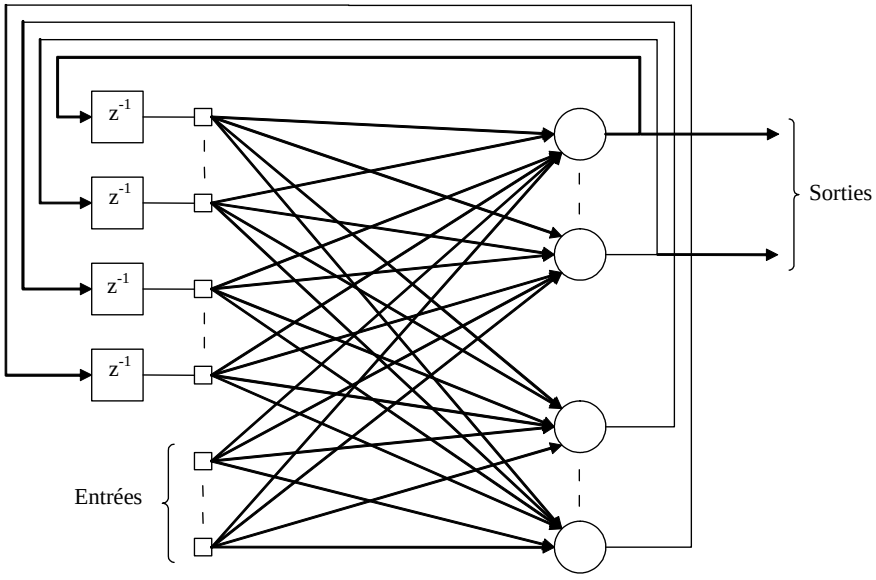
Dans [36], il est décrit un algorithme plus performant qui utilise le filtre de Kalman. Cet algorithme permet l'apprentissage simultané des centres, des largeurs et des poids.

On pourra aussi aller voir [38] pour une étude détaillée des différents algorithmes d'apprentissage.

5.2 Le réseau PRNN (pipelined recurrent neural network)

En 1995, Haykin et Li [41] ont présenté un nouveau prédicteur non linéaire basé sur un réseau de neurone PRNN (pipelined recurrent neural network). L'algorithme d'apprentissage utilisé par Haykin and Li était un algorithme de descente du gradient. Mais les performances de ce prédicteur, en terme de gain de prédiction, était moins bonne que celles d'un simple prédicteur linéaire. Dans [42], Jens Baltersee propose un nouvel algorithme pour le prédicteur non linéaire de Haykin et Li, un algorithme des moindres carrés récursif étendu (ERLS). Gràce à cet algorithme, il surpasse le gain de prédiction d'un prédicteur linéaire RLS de 2dB.

Le réseau de neurones PRNN consiste en la mise en série d'un certain nombre de réseaux RNN (recurrent neural network ou réseau de neurones récurrent).



Le réseau RNN

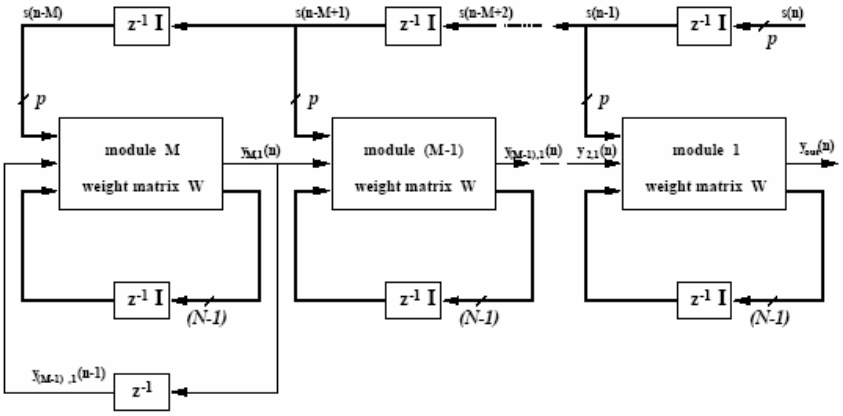
Le réseau de neurones récurrent RNN a la même structure que le perceptron de Rosenblatt ; la différence est que certaine des sorties sont réinjectées avec un retard en entrée du réseau.

La fonction de transfert des neurones de la couche de sortie est une fonction sigmoïde.

Si on note P le nombre d'entrées du réseau, N le nombre de neurones de la couche de sortie et F le nombre de sorties qui sont réinjectées en entrée du réseau (connexions "feedback") alors les paramètres du réseau forme une matrice W de dimension $(P + F + 1) * N$. Les $P + F$ premières

lignes correspondent aux poids associés à chaque neurone et la dernière ligne est le biais de chaque neurone.

Le réseau PRNN décrit dans [41] est la mise en série de M modules RNN identiques, selon la configuration suivante :



Chaque module a N neurones, P entrées,1 sortie et N connexions feedback. Pour les $M-1$ premiers modules, on a enlevé la connexion feedback de la sortie et l'entrée corrsepondante a été reliée à la sortie du module suivant. Tous les modules partagent la même matrice W.

On note $s(n)$ le signal de parole à coder.

Le réseau PRNN est entrainé de telle manière qu'il prédise $s(n)$ à partir des valeurs précédentes $s(n-i)$, $i = 1..M+N$. Pour cela , on met en entrée du module k, les valeurs $s(n-k), s(n-k-1), .., s(n-k-p)$ et on aura en sortie une valeur prédite de $s(n-k+1)$.

On associe alors à chaque module une erreur $e_k(n)$:

$$e_k(n) = s(n-k-1) - y_{k,1}(n)$$

Et on définit la fonction erreur pour le réseau PRNN :

$$E(n) = \sum_{i=1}^{M} \lambda^{i-1} e_i^2(n) \text{ avec } \lambda \in [0,1]$$

L'apprentissage consistera alors à minimiser cette fonction.

Dans [43], on a choisi de mettre à jour les paramètres de la matrice W pour chaque echantillon de $s(n)$.

La procédure de prédiction peut alors être décrite en trois étapes :

1. On calcule les erreurs de prédiction $e_k(n), k = 1..M$ et la fonction erreur $E(n)$.

2. On met à jour à jour la matrice W grâce à un des deux algorithmes décrit dans [43] qui calcule une matrice de correction ΔW à additionner à W.
3. On calcule les nouvelles valeurs d'état du réseau.

En ce qui concerne l'initialisation de la matrice W, on utilise la procédure suivante :

1. On prend les L premiers échantillons du signal s.
2. On calcule L matrices de correction ΔW et on en fait la moyenne : on obtient $\Delta W_{époque}$.
3. On met à jour la matrice W gràce à $\Delta W_{époque}$.

On répète alors 2 et 3 un nombre de fois determiné empiriquement.

Les articles [42] et [43] montrent qu'une prédiction adaptative avec un réseau PRNN donne de meilleurs gains de prédiction qu'une simple prédiction linéaire adaptative.

Un travail qui pourra être envisagé serait donc d'integrer un réseau RBF ou un réseau PRNN dans un codeur audio sans perte et voir si cela donne de meilleurs résultats qu'un codeur qui utilise un réseau MLP.

Conclusion

Le codage audio sans perte est nécessaire lorsqu'on souhaite réduire la taille d'un fichier audio sans perdre d'informations lors de la compression. Pour réaliser ce type de compression, les codeurs existants utilise le schéma : découpage en trames - décorrélation - codage entropique. La majorité des codeurs existants utilise la prédiction linéaire pour décorréler le signal. Or ce type de prédiction a des limites, c'est pourquoi notre étude a porté sur deux autres types de prédiction : une prédiction non linéaire avec un filtre de Volterra et une prédiction non linéaire avec un réseau de neurones.

Nous avons d'abord réalisé des simulations avec Matlab et relevé les conclusions suivantes :

- En approche forward : une prédiction non linéaire stéréo avec un filtre de Volterra et une prédiction non linéaire stéréo avec un réseau de neurones donnent de meilleurs gains de prédiction qu'une prédiction linéaire stéréo.
- Le nombre trés important de coefficients des prédictions non linéaires rendent ce type de prédiction inutilisable dans un codeur audio sans perte en configuration forward car le débit lié aux coefficients est trop important.
- En approche backward : une prédiction non linéaire stéréo avec un réseau de neurones donne de meilleurs gains de prédiction qu'une prédiction linéaire stéréo, mais ce n'est pas le cas pour une prédiction non linéaire stéréo avec un filtre de Volterra.
- Une prédiction mono avec un filtre de Volterra sur les résidus de la prédiction linéaire stéréo ne donne pas de meilleurs gains de prédiction qu'une prédiction linéaire mono , que ce soit dans une approche bacward ou dans une approche adaptative par échantillon.

On en a alors déduit que la seule configuration qui peut être intégrée à un codeur audio sans perte et donner de meilleurs résultats qu'un codeur avec une prédiction linéaire est une prédiction non linéaire stéréo avec un réseau de neurones en backward. Cette configuration a alors été implémentée en C et testée sur des fichiers de musique de style différents. Les ratios obtenus sont supérieurs à ceux obtenus avec un codeur utilisant une prédiction linéaire.

On a pu apporter des améliorations en ajoutant un filtrage NLMS linéaire sur les résidus de la prédiction stéréo afin de décorréler au maximum les signaux. Les ratios obtenus sont comparables à ceux de Monkey Audio, le codeur existant le plus performant.

Cependant, nous n'avons implémenté qu'un seul type de réseau de neurones, le plus simple. Ils en

existent beaucoup d'autres : le réseau RBF, le réseau PRNN... Il serait donc intéressant d'effectuer des simulations avec d'autres types de réseaux afin de voir si ils donnent de meilleurs résultats qu'un réseau de neurones MLP.

Bibliographie

[1] Jean–Luc Garcia, *Codage audio stéréo sans perte*, rapport de stage, fevrier-juillet 2003, groupe de recherche sur la parole et l'audio, université de Sherbrooke.

[2] M. Hans et R. Shafer, *Lossless compression of digital audio*, IEEE Signal Processing magazine, pp 21-32, 2001.

[3] T. Liebchen, M. Purat et P. Noll, *Improved lossless transform coding of audio signals.*

[4] J.D. Johnston, *Perceptual transform of wide band stereo signals*, IEEE, 1989.

[5] J.D. Johnston et A.J. Ferreira, *Sum difference stereo transform coding*, IEEE, 1992.

[6] P. Cambridge et M. Todd, *Audio data compression techniques*, 94th AES convention, 1993.

[7] T. Liebchen, *Lossless audio coding using adaptative multichannel prediction*, 113th AES convention, 2002.

[8] L. Marple, Jr, Digital analysis with applications, pp. 386-421, 1987.

[9] A. Gersho et R. Gray, *Vector Quantization and Signal Compression*, MA : Kluwer Academic, 1992.

[10] T. Robinson, *Simple lossless and near lossless waveform compression*, http ://www-svr.eng.cam.ac.uk/reports/abstracts/robinson tr156.html.

[11] S.W. Golomb, *Run-Length Encodings*, IEEE Trans Info. Theory, Vol 12 pp 399-401 1966.

[12] R. F. Rice, *Some practical universal noiseless coding techniques*, Tech Rep. JPL-79-22, JetPropulsion Laboratory, Pasadena, CA, mars 1979.

[13] J. Thyssen, H. Nielsen and S. D. Hansen, "Non-linear short term prediction in speech coding", In proc. Int. Conf. Acoustics, Speech & signal processing, Australia. Pp.I-185, I-188, April 1994.

[14] J. Thyssen, H. Nielsen and S. D. Hansen, "Quantization of Non-Linear Predictors in Speech Coding", *Proc. ICASSP'95*, pp. 265-268, 1995.

[15] E. Mumolo, D. Francescato, "Adaptive predictive coding of speech by means of volterra predictors", Nonlinear Digital Signal Processing, IEEE Winter Workshop on , Pages :2.1_4.1 - 2.1_4.4, Jan. 17-20, 1993.

[16] J.H. Chen, R.V. Cox, *A Low-Delay CELP Coder for the CCITT 16kb/s Speech Coding Standard*, IEEE J. Select. Areas Commun. vol. 10, No. 5, pp 830-849, juin 1992.

[17] V. J. Mathews, "Adaptive polynomial filters," IEEE Signal Processing Magazine, Vol. 8, No. 3, pp. 10-26, July 1991.

[18] S. Haykin, *Adaptive Filter Theory*, (Englewood Cliffs, NJ : Prnetice Hall, Inc., 1991).

[19] W. S. McCulloch and W. Pitts. *A logical calculus of ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 5 :115–133, 1943.

[20] Rosenblatt, F. (1962). *Principles of Neurodynamics.* New York : Spartan Books.

[21] M. Minsky and S. Papert, Perceptrons, *expanded edition*, MIT Press, 1988.

[22] J.J. Hopfield, "*Neural networks and physical systems with emergent collective computational abilities*," Prot : Nat. Acad. Sci., vol. 79, Apr. 1982, pp. 2554-2558.

[23] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors.* Nature, 323(9) :533–536, October 1986.

[24] http ://www-dapnia.cea.fr/Spp/Experiences/OPAL/bib/nnhistory.html#ref_Rosenblat

[25] M. Faúndez, E. Monte and F. Vallverdú "A comparative study between linear and nonlinear speech prediction". *Biological & artificial computation : from neuroscience to technology, IWANN'97.* Ed. Springer Verlag ISBN 3-540-63047-3, pp.1154-1163

[26] M. Faundez, Francesc Vallverdu & Enric Monte, "Nonlinear prediction with neural nets in ADPCM" ICASSP-98 ., Vol I, pp.345-348. SP11.3 Seattle, USA

[27] M. Faúndez, F. Vallverdú & E. Monte "Efficient nonlinear prediction in ADPCM". 5th International Conference on Electronics, circuits and systems, ICECS´98. Vol.3 pp.543-546. September 1998 Lisboa.

[28] M. Faúndez, O. Oliva "ADPCM with nonlinear prediction". EUSIPCO-98., Rodas pp 1205-1208.

[29] M. Faúndez , "Adaptive Hybrid Speech coding with a MLP/LPC structure". IWANN'99

[30] O. Oliva , M. Faúndez "A comparative study of several ADPCM schemes with linear and nonlinear prediction" EUROSPEECH'99 , Budapest, Vol. 3, pp.1467-1470

[31] M. Faúndez "Nonlinear predictive models computation in adpcm schemes". European signal processing conference, EUSIPCO'2000,September de 2000, Tampere, Vol. II, pp 813-816, ISBN 952-15-0445-5

[32] Gas B, Zarader J.L, Sellem P et Didiot J.C ; " Speech coding by limited weigths Neural Networks " ; International Conference On System, Man and Cybernetics (ICSMC 97'), Orlando, Foride USA, Octobre 1997, vol 5, pp 4081-4086.

[33] Chavy C, Gas B et Zarader J.L ; " Discriminative coding with predictive neural networks " ; International Conference on Artificial Neural Networks (ICANN 99'). September 1999, Edimburg, Scottland, pp 216-220.

[34] Gas B, Zarader J.L , Chavy C ; " A new approach to speech coding : the Neural Predictive Coding " ; Journal of Advanced Computational Intelligence, Vol 4, n°1, Janvier 2001, pp 120-127.

[35] J.L. Zarader, Gas B., Chavy C., Charles Elie Nelson D. ; " New Compression and decompression of speech by a Neural Predictive Coding (NPC)" ;WSES Intern. Conf. in Signal Speech and Image processing (SSIP'01). Advances in Signal Processing and Communications. September 2001, Malta, pp 119-124.

[36] M. Birgmeier, "A fully Kalman-trained radial basis function network for nonlinear speech modeling", in Proc. 1995 IEEE int. Conf. Neural Networks, ICNN'95 Perth, Nov. 1995.

[37] M. Birgmeier "Nonlinear prediction of speech signals using radial basis function networks". EUSIPCO 1996,vol. 1 pag. 459-462.

[38] S. Haykin. *Neural Networks : A Comprehensive Foundation*. Mc Millan College Publishing Co., 1994.

[39] F. Diaz y A. Figueiras "Nonlinear prediction for speech coding using radial basis functions". ICASSP 1995, pp.788-791

[40] F. Diaz de Maria and A. R. Figueiras-Vidal "Radial basis functions for nonlinear prediction of speech in analysis-by-synthesis coders", in Proc. IEEE workshop on nonli. Signal and image processing NSIP'95, Haldiki, June 1995

[41] S. Haykin and L. Li. Non-linear Adaptive Prediction of Non-stationary Signals. *IEEE Transactions on Signal Processing*, 43(2) :526-535,1995.

[42] J. Baltersee and J.A. Chambers, Non-linear Adaptive Prediction of Speech with a Pipelined Recurrent Neural Network and a Linearised Recursive Least Squares Algorithm, *European Conference on Signal Analysis and Prediction (ECSAP-97)*, Prague, Czech Republic, June 24-27, 1997.

[43] D. Mandic, J. Baltersee and J.A. Chambers, Non-linear Adaptive Prediction of Speech with a Pipelined Recurrent Neural Network, Invited chapter in *Signal Analysis and Prediction*, Birkhaueser, Boston, 1998.

[44] M. T. Hagan and M. Menhaj, "Training multilayer networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, 1944, pp. 989-993.

[45] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," *Proceedings of the IJCNN*, vol. 3,pp. 21-26, July 1990.

[46] D. Foresee and M. Hagan, "Gauss-Newton Approximation to Bayesian Learning," *Proceedings of the 1997 International Joint Conference on Neural Networks.*

[47] Bartlett, P.L. (1997), "For valid generalization, the size of the weights is more important than the size of the network," in Mozer, M.C., Jordan, M.I., and Petsche, T., (eds.) *Advances in Neural Information Processing Systems 9*, Cambrideg, MA : The MIT Press, pp. 134-140.