SOMMAIRE

INTRODUCTION 1

CHAPITRE 1
INTRODUCTION AUX RÉSEAUX DE NEURONES 3

1. Introduction 3
2. Historique 4
3. Un neurone artificiel 4
    3.1. Le modèle de McCulloch et Pitts 4
    3.2. Généralisation 6
4. Les réseaux 6
5. La mise à jour des poids 7
6. Apprentissage et adaptation 7
7. Le Perceptron à plusieurs couches 8
    7.1. Description 8
    7.2. Apprentissage par rétropropagation d'erreurs. 10
8. Mise en oeuvre 11
9. Conclusions 12
    9.1. Des propriétés 12
    9.2. Des limites 13
    9.3. Nos prétraitements 13

CHAPITRE 2
LES RESSOURCES MISES EN OEUVRE 15

1. Introduction 15
2. Utilisation de SIRENE 15
3. Le programme 17
    3.1. Présentation 17
    3.2. Création d'un fichier d'instructions type 18
    3.3. Modification du nombre de sorties et fichier d'utilisation 18
    3.4. Analyse des résultats 18
4. Les phases du sommeil 19
    4.1. Description du problème 19
    4.2. Description des données 20
    4.3. Résultats sans compression 20

5. Les caractères manuscrits de la poste allemande 23
5.1. Description du problème et des données 23
5.2. Résultats sans compression 23
6. Les véhicules 24
6.1. Description du problème et des données 24
6.2. Résultats sans compression 25
6.3. Comparaisons avec des résultats officiels 26
7. Données corrélées 28
8. Conclusions 28

CHAPITRE 3
LA MÉTHODE DE KARHUNEN-LOÈVE 29

1. Introduction 29
2. La méthode 29
2.1. Description formelle 29
2.2. Signification géométrique 32
3. Le programme 34
3.1. Vecteurs propres 34
3.2. Matrice de transformation 35
3.3. Création du fichier compressé 35
4. Analyse des phases du sommeil 35
4.1. Introduction 35
4.2. Compression à 20 36
4.3. Compression à 10 38
5. Reconnaissance des caractères manuscrits 40
5.1. Introduction 40
5.2. Compression à 40 et recentrage 40
5.3. Compression à 20 et recentrage 43
5.4. Autres tentatives de compression 45
5.5. Comparaisons avec des résultats officiels 46
6. Reconnaissance des véhicules 47
6.1. Introduction 47
6.2. Compression à 10. 47
6.3. Compressions inférieures à 10 49
7. Conclusions 50

CHAPITRE 4
LA MÉTHODE LPC 52

1. Introduction 52
2. La méthode 52
3. Le programme 57
4. Analyse des phases du sommeil 57
    4.1. Introduction 57
    4.2. Compression à 20 57
    4.3. Compression à 10 60
5. Reconnaissance des caractères manuscrits 62
    5.1. Introduction 62
    5.2. Compression à 40 62
    5.3. Autres tentatives 64
6. Reconnaissance des véhicules 65
7. Conclusions 66

CHAPITRE 5
LA MÉTHODE NLPCA 67

1. Introduction 67
2. La méthode 67
    2.1. Le principe 67
    2.2. Recherche des vecteurs de fonctions G et H 68
    2.3. L'apprentissage des réseaux 70
    2.4. Détermination de la taille des couches 72
    2.5. Remarques sur les couches cachées 73
3. Le programme 73
    3.1. Conversion des données au format NLPCA. 74
    3.2. Sélection des premières couches d'un réseau. 74
    3.3. Modification du nombre de sorties et utilisation type 74
    3.4. Récupération des sorties d'un réseau 75
4. Les traitements 75
5. Les phases du sommeil et les caractères manuscrits 77
6. Reconnaissance des véhicules 77
    6.1. Introduction 77
    6.2. Compression à 10 78
    6.3. Compression à 8 81
    6.4. Autres compressions 83

7. Données corrélées 84
7.1. Présentation 84
7.2. Résultats 85
8. Conclusions 85

CHAPITRE 6
LA MÉTHODE LSP 87

1. Introduction 87
2. La méthode 87
3. Objections à son implémentation 89
4. Conclusions 90

CHAPITRE 7
COMPARAISONS DES MÉTHODES 91

1. Introduction 91
2. Les résultats 91
3. Conclusions 94

CONCLUSIONS 95

BIBLIOGRAPHIE

ANNEXES
A. Les menus de SIRENE
B. Les menus de notre programme
C. Code source du programme

REMERCIEMENTS

Avant de présenter notre travail, nous tenons à remercier tous ceux qui, de loin ou de près, ont participé à sa réalisation.

Nos remerciements vont tout d'abord à Monsieur Destiné qui a patronné ce travail et à Monsieur Fombellida qui nous a conseillé et guidé.

Nous tenons également à exprimer notre reconnaissance à Mme Haesbroeck pour la relecture minutieuse de notre texte et les conseils qu'elle nous a donnés.

Beaucoup d'autres personnes devraient être mentionnées ici. Citons entre autres : notre famille, Gentiane et ses parents, les habitués de la salle IA, ainsi que tous les utilisateurs du réseau d'ordinateurs qui n'ont pas protesté contre la surcharge de leur machine durant nos nombreuses simulations neurales.

UNIVERSITÉ DE LIÈGE
Faculté des Sciences Appliquées

PRÉTRAITEMENT DE DONNÉES EN RECONNAISSANCE DE FORMES PAR RNA

Travail de fin d'études présenté par

Michaël SCHYNS

en vue de l'obtention du grade de
Licencié en Informatique
Année académique 1992-1993

SOMMAIRE

INTRODUCTION

CHAPITRE 1

Introduction aux réseaux de neurones

1. Introduction
2. Historique
3. Un neurone artificiel
    3.1. Le modèle de McCulloch et Pitts
    3.2. Généralisation
4. Les réseaux
5. La mise à jour des poids
6. Apprentissage et adaptation
7. Le Perceptron à plusieurs couches
    7.1. Description
    7.2. Apprentissage par rétropropagation d'erreurs
8. Mise en oeuvre
9. Conclusions
    9.1. Des propriétés
    9.2. Des limites
    9.3. Nos prétraitements

CHAPITRE 2

Les ressources mises en oeuvre

1. Introduction
2. Utilisation de SIRENE
3. Le programme
    - 3.1. Présentation
    - 3.2. Création d'un fichier d'instructions type
    - 3.3. Modification du nombre de sorties et fichier d'utilisation
    - 3.4. Analyse des résultats
4. Les phases du sommeil
    - 4.1. Description du problème
    - 4.2. Description des données
    - 4.3. Résultats sans compression
5. Les caractères de la poste allemande
    - 5.1. Description du problème et des données
    - 5.2. Résultats sans compression
6. Les véhicules
    - 6.1. Description du problème et des données
    - 6.2. Résultats sans compression
    - 6.3. Comparaisons avec des résultats officiels
7. Données corrélées
8. Conclusions

CHAPITRE 3

La méthode de Karhunen-Loève

1. Introduction
2. La méthode
    2.1. Description formelle
    2.2. Signification géométrique
3. Le programme
    3.1. Vecteurs propres
    3.2. Matrice de transformation
    3.3. Création du fichier compressé
4. Analyse des phases du sommeil
    4.1. Introduction
    4.2. Compression à 20
    4.3. Compression à 10
5. Reconnaissance des caractères manuscrits
    5.1. Introduction
    5.2. Compression à 40 et recentrage
    5.3. Compression à 20 et recentrage
    5.4. Autres tentatives de compression
    5.5. Comparaisons avec des résultats officiels
6. Reconnaissance des véhicules
    6.1. Introduction
    6.2. Compression à 10
    6.3. Compressions inférieures à 10
7. Conclusions

CHAPITRE 4

La méthode LPC
(Linear Predictive Coding)

1. Introduction
2. La méthode
3. Le programme
4. Analyse des phases du sommeil
    4.1. Introduction
    4.2. Compression à 20
    4.3. Compression à 10
5. Reconnaissance des caractères manuscrits
    5.1. Introduction
    5.2. Compression à 40
    5.3. Autres tentatives
6. Reconnaissance des véhicules
7. Conclusions

CHAPITRE 5

La méthode NLPCA
(NonLinear Principal Component Analysis)

1. Introduction
2. La méthode
    2.1. Le principe
    2.2. Recherche des vecteurs de fonctions G et H
    2.3. L'apprentissage des réseaux
    2.4. Détermination de la taille des couches
    2.5. Remarques sur les couches cachées
3. Le programme
    3.1. Conversion des données au format NLPCA
    3.2. Sélection des premières couches d'un réseau
    3.3. Modification du nombre de sorties et utilisation type
    3.4. Récupération des sorties d'un réseau
4. Les traitements
5. Les phases du sommeil et les caractères manuscrits
6. Reconnaissance des véhicules
    6.1. Introduction
    6.2. Compression à 10
    6.3. Compression à 8
    6.4. Autres compressions
7. Données corrélées
    7.1. Présentation
    7.2. Résultats
8. Conclusions

CHAPITRE 6

La méthode LSP
(Line Spectral Pair)

1. Introduction
2. La méthode
3. Objections à son implémentation
4. Conclusions

CHAPITRE 7

Comparaisons des méthodes

1. Introduction
2. Les résultats
3. Conclusions

CONCLUSIONS

BIBLIOGRAPHIE

ANNEXES

A. Présentation des menus de SIRENE

B. Présentation des menus de notre programme

C. Code source de notre programme

INTRODUCTION

Depuis des siècles, les hommes développent des machines pour simplifier leur vie. Le début de cette ère de construction commença avec la découverte de machines simples telles que le levier, la roue et la poulie. De nos jours, ingénieurs et scientifiques essayent de développer des machines intelligentes. Parmi elles, on trouve les réseaux de neurones artificiels (RNA).

Les hommes et les animaux sont bien meilleurs et plus rapides pour reconnaître des images que le plus avancé des ordinateurs. Les réseaux de neurones artificiels (RNA) tentent d'imiter leur comportement c'est-à-dire apprendre par expériences et être ensuite capable de prendre rapidement de bonnes décisions. Ces capacités sont basées sur le fait que nous pouvons reproduire certaines des caractéristiques du cerveau humain à l'aide de moyens artificiels. Un réseau de neurones est réalisé par un maillage de noeuds fonctionnels, appelés neurones, et de connexions entre eux. Ils opèrent collectivement et simultanément sur la plus grande partie ou sur toutes les données et entrées. Nous présenterons dans ce travail le modèle le plus couramment utilisé et déterminerons ses nombreuses qualités.

Les réseaux de neurones n'ont malheureusement pas que des avantages. Leur taille croît avec la quantité et la complexité des données à traiter. Or le principal inconvénient du RNA est lié à sa complexité. Plus elle est grande, plus le RNA sera difficile et coûteux à implémenter physiquement et plus son temps d'apprentissage sera grand. Pour résoudre ce problème, une solution est de prétraiter les données pour diminuer leur taille.

Il nous a dès lors été demandé d'analyser trois méthodes de compression applicables au traitement par RNA : la méthode de Karhunen-Loève, la méthode LPC (Linear Predictive Coding) et la méthode NLPCA (Non Linear Principal Composant Analysis). Nous en avons ajouté une : la méthode LSP (Line Spectral Pair). Les chapitres qui y sont consacrés tenteront de convaincre le lecteur de l'efficacité de ces procédés. Nous déterminerons également comment choisir la méthode à utiliser pour un problème de classification posé à un RNA. Pour ce faire, deux analyses sont réalisées. La première consiste en un développement théorique de la méthode. La seconde, plus importante, est son application à des ensembles de données représentatifs, correspondant à des problèmes commerciaux et industriels complexes. On analysera

ensuite les résultats de classification d'un réseau de neurones à partir de ces données prétraitées. Il y aura également des comparaisons avec les résultats sans prétraitement.

Trois problèmes de reconnaissance de formes seront envisagés. Le premier demande à un RNA de reconnaître des phases du sommeil à partir d'enregistrements polygraphiques. Le second est un problème de reconnaissance de caractères manuscrits digitalisés. Le dernier est un problème de distinction de silhouettes de véhicules. Ces données ont été choisies pour mettre en valeur certains prétraitements par rapport aux autres. De plus, l'utilisation d'exemples réels montre que les réseaux de neurones sont déjà applicables. Ce travail prouve leur utilité et leur efficacité au niveau commercial et industriel.

Nous espérons vous convaincre de l'utilité des réseaux de neurones et des prétraitements et nous vous souhaitons une bonne lecture.

CHAPITRE 1
INTRODUCTION AUX RÉSEAUX DE NEURONES

1. Introduction

Différents modèles de réseaux de neurones ont déjà été présentés. Leur caractéristique commune est de vouloir imiter certaines des propriétés du cerveau humain en reproduisant une partie de ses structures élémentaires.

Dans l'état actuel de nos connaissances dans le domaine de l'intelligence artificielle et plus généralement de l'informatique, nous ne disposons pas toujours d'algorithmes efficaces pour résoudre des problèmes complexes tels que :

- permettre à un robot de conduire un véhicule dans un environnement variable.
- réaliser la lecture automatique d'un texte manuscrit produit par n'importe quelle personne.
- reconnaître la parole quelle que soit la personne qui parle.
- reconnaître des visages d'individu indépendamment de l'angle sous lequel ils sont présentés.
- ...

Les ordinateurs sont extrêmement rapides et précis pour exécuter des séquences d'instructions qui ont été formulées pour eux. Malheureusement, les problèmes cités nécessitent la considération simultanée d'un très grand nombre de contraintes, parfois mal définies. Il est donc difficile de leur trouver une formulation informatique. De plus, le système de traitement humain de l'information est composé de neurones qui travaillent à des vitesses à peu près un million de fois plus lentes que les circuits d'un ordinateur. Cependant, les humains sont beaucoup plus efficaces pour résoudre des problèmes complexes comme ceux mentionnés plus haut. L'organisation du cerveau humain semble donc une des clefs du problème. Malheureusement, la compréhension des systèmes neuraux biologiques n'est pas encore très avancée. Nous devons donc nous limiter à des modèles très simplifiés. On peut dès lors supposer que

cette technique ne pourra que se développer avec l'amélioration de nos connaissances sur le cerveau.

Cette capacité de traitement théorique et les premiers résultats obtenus dans la pratique méritent dès lors que l'on s'intéresse aux réseaux de neurones. Nous décrirons donc brièvement un modèle généralement utilisé : le perceptron à plusieurs couches. Nous en déduirons quelques autres avantages des réseaux de neurones par rapport à la programmation traditionnelle, mais aussi certains inconvénients. Cette étude permettra de comprendre la nécessité de méthodes de prétraitements et des conditions de leur implémentation, sujet de ce travail.

2. Historique

La première modélisation d'un neurone date de 1943. Elle a été présentée par McCulloch et Pitts. L'interconnexion de ces neurones permet le calcul de plusieurs fonctions logiques. En 1949, Hebb propose le premier mécanisme d'évolution des connections, appelées par analogie des synapses. L'association de ces deux méthodes permit à Rosenblatt en 1958 de décrire le premier modèle opérationnel de réseaux de neurones : le perceptron. Celui-ci est capable d'apprendre à calculer un grand nombre de fonctions booléennes, mais pas toutes. Ses limites théoriques furent mises en évidence par Minsky et Papert en 1969. Depuis 1985, de nouveaux modèles mathématiques ont permis de les dépasser. Cela a donné naissance au perceptron multicouches que nous étudierons plus particulièrement.

3. Un neurone artificiel

3.1. Le modèle de McCulloch et Pitts

La première définition formelle d'un neurone artificiel basée sur le modèle biologique a été formulée par McCulloch et Pitts. Une représentation en est donnée à la figure 1.1a. Les entrées X_i (i=1, 2, ..., n) sont booléennes (présence ou absence d'impulsion à l'instant k). La sortie est identifiée par le symbole O. W_i est le poids associé à la connexion. La fonction calculée par le neurone est définie comme suit :

$$O^{k+1} = \begin{cases} 1 & si \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & si \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

Notons qu'un poids négatif inhibe une connexion, au contraire d'un poids positif qui la renforce. C'est le choix des valeurs de ces poids qui permettra de réaliser la fonction recherchée. La figure 1.1 montre comment on peut construire des portes élémentaires grâce à ces "coefficients synaptiques". On peut en déduire que ce modèle simpliste permet déjà la réalisation d'un ordinateur digital de complexité arbitraire.

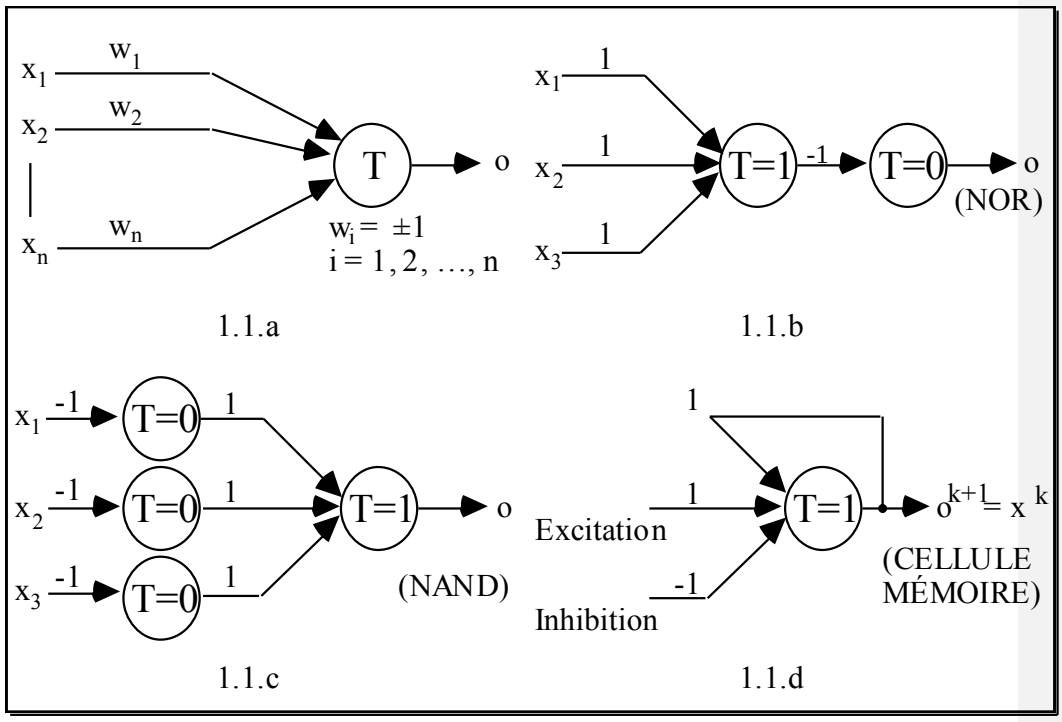


Figure 1.1

3.2. Généralisation

Le précédent modèle a plusieurs inconvénients. Il est binaire et statique. Les coefficients et les seuils θ sont fixés définitivement. Il est assez simple de résoudre ces problèmes. La figure 1.2 représente la nouvelle forme du neurone.

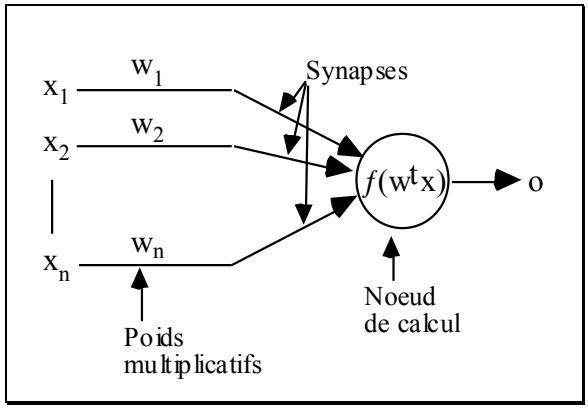


Figure 1.2

Chaque neurone consiste en un élément de traitement (processeur) de plusieurs entrées et calculant une seule sortie. Les poids synaptiques sont représentés par un vecteur W dont les éléments sont modifiables. La sortie est calculée par une *fonction d'activation* f. Différentes fonctions sont envisageables. Les plus courantes sont la fonction sigmoïde et la fonction signe. Notons que le seuil du modèle de McCulloch et Pitts est implicitement représenté par une connexion dont le poids est -1 et l'entrée θ.

4. Les réseaux

Une fois la structure d'un neurone établie, la définition d'un réseau est immédiate : un réseau de neurones est une interconnexion de neurones telle que leur sortie est connectée, avec un poids synaptique, aux entrées d'autres neurones. Le choix des neurones connectés entre eux détermine l'architecture du réseau. Différents modèles existent : multicouches, Hopfield, Kohonen, Boltzmann. Chacun a ses propres techniques d'apprentissage et s'applique plus particulièrement à certains domaines.

Disposant d'une bonne méthode d'apprentissage pour les réseaux multicouches et ceux-ci répondant à nos besoins, nous laisserons le loisir au lecteur intéressé de consulter les ouvrages de références pour les autres modèles.

5. La mise à jour des poids

Il existe essentiellement trois modes possibles pour l'évolution de l'état du réseau :

a) Mode séquentiel : les neurones réévaluent leur sortie l'un après l'autre dans un ordre déterminé.
b) Mode aléatoire asynchrone : chaque neurone réévalue sa sortie à des intervalles de temps aléatoires.
c) Mode parallèle : tous les neurones réévaluent leur sortie périodiquement et simultanément.

6. Apprentissage et adaptation

L'apprentissage consiste à trouver le réseau qui fourni la meilleure solution au problème qu'il doit résoudre, pour un ensemble de données dites d'apprentissage. Cela est réalisé en adaptant, grâce à certaines règles, les vecteurs de poids. Ce concept d'apprentissage n'est vraiment intéressant que si le réseau possède des capacités de généralisation, c'est-à-dire qu'il est capable de fournir une bonne solution pour d'autres données.

De nombreuses règles d'adaptation des poids existent : loi de Hebb (renforcer une connexion entre deux neurones actifs), règle du Perceptron (modifier les poids en fonction de l'erreur entre la sortie souhaitée et celle obtenue), loi delta, de Widrow-Hoff, de corrélation, ou encore du type "le gagnant prend tout". Plus généralement, on peut répartir ces méthodes d'adaptation en deux classes :

1) *Les apprentissages supervisés ou avec professeur*. Dans cette catégorie, on suppose que chaque fois qu'une entrée est appliquée au réseau, la sortie désirée est fournie par le professeur. Celui-ci pourra alors récompenser ou punir le réseau selon l'erreur commise en ajustant les poids. Cette grande classe

d'algorithmes peut se subdiviser en trois sous-catégories : auto-association, hétéro-association et classification.

2) *Les apprentissages non supervisés.* Dans cette catégorie, la réponse désirée est inconnue. Le réseau est alors supposé découvrir lui-même la meilleure représentation de l'information fournie.

Il est important de remarquer que c'est l'ensemble des poids ainsi obtenus qui détermine le réseau résolvant le problème posé. La dégradation accidentelle d'un de ces poids n'aura en général que peu d'influence sur le résultat final; les autres assurant la convergence vers la solution.

7. Le Perceptron à plusieurs couches

7.1. Description

Ce type de Perceptron est un réseau multicouches non récurrent. Sa structure est définie par :

a) L'ensemble des neurones du réseau peut être divisé en $N \geq 3$ sous-ensembles disjoints. Chacun de ces ensembles s'appelle une couche.

b) Il existe une numérotation de 1 à N de ces couches telle que :

- deux neurones appartenant à des couches différentes ne peuvent être directement connectés que si ces deux couches sont adjacentes; c'est-à-dire si leur indice diffère d'une unité.
- la couche 1 s'appelle la couche d'entrée et est la seule dont les neurones ont leur état directement influencé par l'environnement. En réalité, ces neurones sont fictifs et se contentent de jouer le rôle d'interface entre le réseau et l'environnement.
- la couche N s'appelle la couche de sortie et est la seule qui fournisse directement une réponse à l'environnement.
- les autres couches sont appelées couches cachées et ne peuvent communiquer directement avec l'environnement.

- les réseaux non récurrents ne possèdent pas de cycle; c'est-à-dire que les connexions se font toujours d'un neurone vers un autre de la couche immédiatement supérieure.

Un exemple d'un tel réseau est représenté à la figure 1.3. Ici, l'information progresse de couche en couche en partant de celle d'entrée pour arriver à celle de sortie où elle représente alors la réponse fournie par le système à l'environnement.

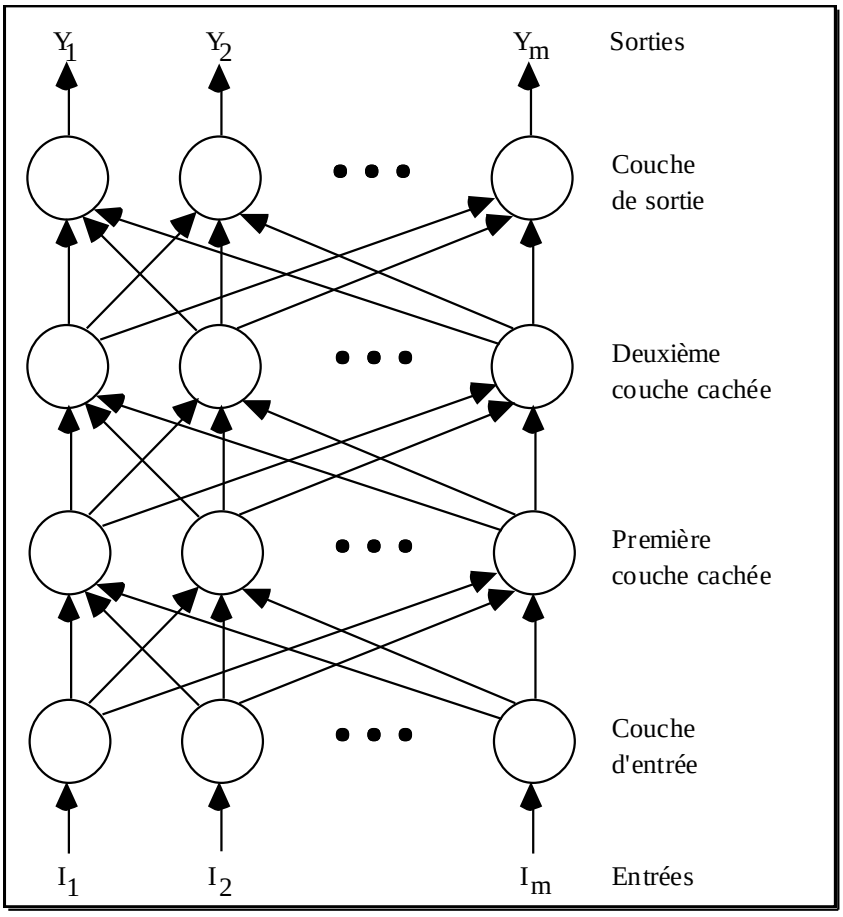


Figure 1.3

Il peut être montré que pour n'importe quel problème de classification, il existe un réseau non récurrent à trois couches qui résout ce problème. Cependant, il n'est pas sans intérêt d'utiliser un plus grand nombre de couches intermédiaires. En effet, cet accroissement du nombre de couches s'accompagne généralement d'une

amélioration des capacités de généralisation, d'une plus grande résistance aux dommages et d'une meilleure efficacité quant à la représentation interne construite par le réseau pour stocker les données lors de l'apprentissage.

7.2. Apprentissage par rétropropagation d'erreurs.

L'algorithme de la rétropropagation d'erreurs est actuellement la procédure d'apprentissage la plus utilisée. Cette popularité est une conséquence des résultats généralement bons qu'elle permet d'obtenir pour un grand nombre de problèmes différents et cela en réalisant des simulations sur machines séquentielles, en des temps raisonnables vis-à-vis de ceux requis par d'autres algorithmes tel que, par exemple, celui du recuit simulé. Cette règle est en fait une généralisation de la loi delta et pour cette raison est souvent appelée loi delta généralisée.

Dans cet algorithme, de même que l'on est capable de propager un signal provenant des neurones d'entrée, on peut, en suivant le chemin inverse, rétropropager l'erreur commise en sortie vers les couches internes et modifier en conséquence les poids. Cette minimisation de l'erreur permet de mémoriser la relation définie entre l'entrée et la sortie désirée. Ce schéma est appliqué à chaque paire entrée-sortie de l'ensemble d'apprentissage et recommencé plusieurs fois jusqu'à l'obtention d'une erreur globale acceptable. Le nombre d'itérations nécessaires n'est malheureusement pas estimable.

Nous donnons ici une formalisation sommaire de cet algorithme. Pour un exemple à apprendre donné, on note X le vecteur des entrées et Y le vecteur des sorties désirées. Si le réseau comporte n neurones en entrée et m en sortie, on a donc :

$X = (X_1, X_2, \ldots, X_n)$ et $Y = (Y_1, Y_2, \ldots, Y_m)$

On note $S = (S_1, S_2, \ldots, S_m)$ le vecteur des sorties obtenues à l'issue de la propagation avant de l'exemple X dans le réseau. On cherche à minimiser l'erreur quadratique entre les sorties désirées et les sorties obtenues, cette erreur étant considérée comme une fonction des poids des connexions :

$$E(w) = \sum_{i=1}^{m} (Y_i - S_i)^2$$

La règle de modification des poids à la présentation numéro k de l'exemple X est :

$$W_{ij}(k) = W_{ij}(k-1) - e(k) \cdot d_j \cdot O_j$$

où d_i est calculé de proche en proche de la couche de sortie à la couche d'entrée :

(1) $d_i = 2 \cdot (S_i - Y_i) \cdot f'(I_i)$ pour la couche de sortie
(2) $d_i = \sum_h d_h \cdot W_{hi} \cdot f'(I_i)$ pour les couches cachées.

où h parcourt les neurones vers lesquels le neurone i envoie une connexion.

f est une fonction d'activation dérivable (sigmoïde).
O_j est la sortie du neurone j.
I_i est l'entrée du neurone i, $I_i = \sum_j W_{ij} \cdot O_j$
e(k) est le pas du gradient à l'étape k.

Cet algorithme a l'avantage d'être local, c'est-à-dire que la plupart des calculs d'apprentissage peuvent être effectués au niveau de chaque neurone indépendamment (avec un minimum de contrôle global).

Une règle à respecter pour faciliter la convergence est d'utiliser un ensemble d'apprentissage contenant au moins dix fois plus de vecteurs de données, souvent appelés patterns, que de synapses dans le réseau.

8. Mise en oeuvre

Les réseaux de neurones sont actuellement réalisés de nombreuses manières différentes. Pour ce travail, nous utilisons un simulateur sur un ordinateur séquentiel conventionnel. Cependant, pour obtenir les meilleurs résultats, il est nécessaire de les implémenter physiquement en tenant compte qu'ils sont massivement parallèles : chaque neurone peut être vu comme un processeur indépendant, aux fonctions très simples. Le problème auquel on est confronté actuellement est celui des connexions. Il est très coûteux de réaliser un circuit à très forte connectivité en VLSI. Cependant, des études montrent que des implémentations électroniques des réseaux sont réalisables et prometteuses.

En attendant les résultats de ce développement technique, on utilise des simulateurs sur des ordinateurs digitaux. Ces derniers sont généralement dénommés neuro-ordinateurs programmables. Les plus performants étant bien entendu ceux qui peuvent travailler en parallèle. Malgré cela et d'autres améliorations comme des cartes accélératrices, ces neuro-ordinateurs ne peuvent rivaliser avec une implémentation spécifique.

9. Conclusions

9.1. Des propriétés

L'intérêt porté aujourd'hui aux réseaux de neurones tient sa justification dans les quelques propriétés fascinantes qu'ils possèdent. Citons les plus importantes : le parallélisme, la capacité d'adaptation, la mémoire distribuée, la capacité de généralisation et sa facilité de construction.

Le parallélisme se situe à la base de l'architecture des réseaux de neurones considérés comme ensembles d'entités élémentaires qui travaillent simultanément. Le parallélisme permet une rapidité de calcul supérieure, mais exige de penser et de poser différemment les problèmes à résoudre.

La capacité d'adaptation permet au réseau de tenir compte de nouvelles contraintes ou de nouvelles données du monde extérieur. Cette capacité présente un intérêt déterminant pour tous les problèmes évolutifs. Il faut, pour les résoudre, pouvoir tenir compte de situations non encore connues.

Dans les réseaux de neurones, la "mémoire" d'un fait correspond à une carte d'activation de l'ensemble des neurones. Cela permet une meilleure résistance au bruit. La perte d'un élément ne correspond pas à la perte d'un fait mémorisé, mais à une dégradation, d'autant plus faible qu'il y a de synapses. De plus, la recherche d'un fait ne nécessite pas la connaissance de l'endroit de stockage, le réseau entier se chargeant de le restituer.

La capacité de généralisation est essentielle. Elle assure que le réseau donnera une bonne solution pour une entrée ne figurant pas dans l'ensemble d'apprentissage. Nombre de problèmes résolus par des experts le sont de façon plus ou moins intuitive, ce qui rend difficile l'exposé explicite des connaissances et des règles nécessaires à leur solution. Le réseau adopte une démarche semblable à celle de l'expert.

Le principe des réseaux de neurones et leur structure sont assez simples. La simulation informatique ne nécessite qu'un temps de développement assez court.

9.2. Des limites

Un des principaux reproches fait aux réseaux de neurones est l'impossibilité d'expliquer les résultats qu'ils fournissent. Les réseaux se présentent comme des boîtes noires dont les règles de fonctionnement sont inconnues. Ils créent eux-mêmes leur représentation lors de l'apprentissage. La qualité de leurs performances ne peut être mesurée que par des méthodes statistiques, ce qui amène parfois une certaine méfiance de la part des utilisateurs potentiels.

Le second problème a déjà été invoqué : la mise en oeuvre physique. Les réseaux de neurones seront optimums quand ils auront leur propre support. Différentes solutions ont été envisagées pour faciliter ce problème; notamment diminuer le nombre de neurones (par complexification de leur structure, par prétraitements…).

9.3. Nos prétraitements

Étant donné les qualités des réseaux de neurones, il est intéressant d'essayer de supprimer un maximum de leurs défauts. Nous présenterons dans ce travail quelques idées pour diminuer la complexité des réseaux.

Nos prétraitements visent à réduire la taille d'un pattern d'entrée et donc la taille du réseau. L'importance d'une telle démarche découle de ce qui vient d'être dit. Outre le fait qu'un réseau avec peu de connexions sera plus facilement implémentable et à meilleur marché qu'un réseau complexe, on obtient surtout un gain appréciable de traitement. Un exemple illustrera bien ce point. Le premier problème que nous avons

du manipuler dans ce travail est la classification de signaux physiologiques en 3 classes. Initialement, un réseau de 3 couches était utilisé : 100 neurones en entrée, 20 en couche cachée et 3 en sortie. On dispose d'un ensemble d'apprentissage de 5 400 patterns. D'après l'algorithme de la rétropropagation, on doit donc évaluer pour chaque pattern 123 fonctions lors de la propagation et 2060 lors de la rétropropagation (100 * 20 + 20 * 3 synapses). Chaque itération nécessite donc près de 12 millions d'évaluations. Celle-ci étant reproduite jusqu'à minimisation de l'erreur. Aussi puissant que soit notre ordinateur séquentiel, la consommation de CPU et de temps a été énorme, inacceptable. A l'aide d'un prétraitement, la taille des patterns d'entrée a été ramenée de 100 à 10. Le réseau choisi comprenait 10 neurones en couche d'entrée, 5 en couche cachée et 3 en sortie. Chaque itération de 5 400 patterns ne nécessite plus que 450 000 évaluations. De plus, l'apprentissage est facilité car maintenant on dispose d'un ensemble d'apprentissage 83 fois plus important que le nombre de synapses. Le nombre d'itérations permettant la convergence en est diminué.

Ce premier chapitre visait à montrer l'intérêt des réseaux de neurones. On déduit également de cette brève présentation, l'importance de prétraitements à réaliser. C'est pourquoi quatre méthodes seront décrites et analysées dans les chapitres suivants.

CHAPITRE 2
LES RESSOURCES MISES EN OEUVRE

1. Introduction

Il nous a été demandé d'analyser les résultats de trois méthodes de compression : Karhunen-Loève, Linear Predictive code (LPC) et NLPCA. La partie la plus importante de notre travail consistait à tester la qualité de ces prétraitements sur différents types et ensembles de données. Nous avons comparé les capacités de réseaux de neurones à reconnaître ces données sans et avec prétraitements.

Ce second chapitre a pour objectif de présenter les ressources mises en oeuvre : les données, les programmes et les ordinateurs. Il contient également les principaux résultats concernant la classification des réseaux de neurones sans prétraitement. Chacun des trois chapitres suivants étudiera les qualités d'une méthode de compression sur les mêmes ensembles de données.

2. Utilisation de SIRENE

Tous nos prétraitements ont pour objectif de fournir des ensembles de données utilisables, plus facilement, par un réseau de neurones. Pour tester la qualité de nos compressions, nous devons comparer les résultats du réseau sans et avec prétraitements. Pour obtenir ces résultats, nous avons utilisé le programme SIRENE[1]. Nous disposions de la version 1 release 9. On trouvera en annexe une présentation des menus principaux de ce programme.

La première étape est la définition du réseau; c'est-à-dire son type, le nombre de couches, le nombre de neurones par couche, les fonctions des neurones, l'algorithme d'apprentissage, les critères de réussite et un ensemble de petits paramètres contrôlant l'apprentissage. La table 2.1 indique un choix que nous avons couramment pratiqué. Ces définitions ne sont pas innées. C'est le problème des réseaux de

[1] SIRENE est un SImulateur de REseaux de NEurones. Il a été écrit par M. Fombellida dans le cadre de sa thèse de doctorat à l'université de Liège, service de Micro-électronique.

neurones : on ne peut pas dire à priori quelle est la meilleure configuration. Nous avons donc procédé à différents essais avant d'obtenir de bons résultats (les meilleurs ?).

La seconde étape est la définition des données à traiter. Nous fournissions trois fichiers de données à SIRENE. Le premier, le plus gros, est le fichier d'apprentissage. Le réseau l'utilise pour modifier ses poids. Le second, le plus petit, est le fichier de validation croisée. Il suit l'apprentissage et est utilisé par le réseau pour détecter un sur-apprentissage c'est-à-dire le moment où le réseau se spécialiserait trop : il étudie les données de l'apprentissage et n'est plus capable de généraliser à d'autres exemples. C'est ce critère qui détermine le meilleur réseau à conserver. Le dernier fichier est un ensemble test; il n'est pas utilisé pour entraîner le réseau. Après chaque étape de mise à jour des poids, il est présenté à l'entrée du réseau et on calcule les sorties que l'on obtiendrait si on arrêtait l'apprentissage à ce moment. Il permet de connaître le comportement du réseau pour un fichier de données quelconque (il est clair que les résultats sont meilleurs pour le fichier qui a servi à l'apprentissage). Notons qu'il est préférable que les fichiers d'apprentissage et de validation croisée n'aient pas de parties communes; cela fausserait le critère de sur-apprentissage. De plus, comme nous le montrerons plus loin, il a été parfois nécessaire de prétraiter ces fichiers pour en recentrer les données dans l'intervalle [-1,1].

Type de réseau :	Perceptron multicouches
Nombre de couches :	3
Nombre de neurones :	En fonction du problème
Fonction d'un neurone :	Tangente hyperbolique
Algorithme d'apprentissage :	Quasi-Newton
Critère de succès :	Small composite error
Fonction d'apprentissage :	Total sum squared error + min. weights
Fonction de validation :	Classification error (%) (Maximum criteria)
Fonction de test :	Classification error (%) (Maximum criteria)
Paramètres :	Overlearning detection and backtracking

Table 2.1

La structure du réseau et les données à traiter connues et placées dans un fichier d'instructions, nous pouvions exécuter le programme SIRENE. Nos simulations ont tourné sur différentes machines SUN. Quatre fichiers sont générés. Trois concernent les résultats de l'apprentissage et le quatrième contient la définition du

réseau qui a donné les meilleurs résultats jusqu'à ce moment. Étant donné le temps d'apprentissage le plus souvent très long, les simulations étaient lancées en tâches de fond. Une fois le job en cours, nous n'avions donc plus de contrôle si ce n'est sa suppression radicale. Il ne restait qu'à espérer que le réseau sauve des (bons) résultats, car nous ne pouvions plus l'y forcer. Nous devions également espérer que la machine utilisée ne soit pas réinitialisée, comme cela a été plusieurs fois le cas. En même temps que SIRENE, nous utilisions la commande TIME pour connaître les différents temps d'exécution de la simulation. Il est important de signaler qu'étant donné que nous n'étions pas les seuls à utiliser les machines SUN et que SIRENE est gourmand en CPU, nos simulations tournaient avec un degré de priorité inférieur (nice). Le temps réel d'exécution en est augmenté.

Enfin, une fois l'apprentissage réalisé, les ensembles de données sont à nouveau présenté au réseau et SIRENE calcule les résultats pour chaque pattern; c'est-à-dire les sorties obtenues, les erreurs sur ces sorties et le résultat de la classification (correcte ou incorrecte). Il reste à analyser ces résultats.

3. Le programme

3.1. Présentation

Notre programme est composé de quatre parties. Trois concernent les méthodes de compression et seront développées dans les chapitres respectifs. La quatrième partie est un module de fonctions complémentaires pour SIRENE. Elles ne sont pas indispensables, mais simplifient souvent la vie à l'utilisateur. Dans ce travail, elles étaient d'une grande utilité. En annexe, nous présentons tous les menus du programme.

```
+++ OUTILS POUR L'UTILISATION DE SIRENE +++


0. Aide


1. Creation d'un fichier d'instructions type
2. Modification du nombre de sorties d'un fichier de donnees
3. Utilisation type d'un reseau de SIRENE
```

```
4. Analyse des resultats

9. Retour au menu principal
```

3.2. Création d'un fichier d'instructions type

Les paramètres de configuration de la table 2.1 ont toujours donné de bons résultats. Dans notre cas, nous avons utilisé SIRENE plus de quarante fois et la plupart du temps avec ces mêmes paramètres. Pour ne pas les redéfinir à chaque utilisation et ne pas devoir passer par les nombreux sous-menus de SIRENE, le sous-progamme 1 crée un fichier d'instructions type les contenant, ainsi que d'autres données nécessaires à la bonne exécution de SIRENE. L'utilisateur doit juste entrer le nom des fichiers de données. Le résultat est un fichier texte nommé "sirene.instr". Pour l'utiliser, il suffit de faire une redirection d'entrées : *sirene <sirene.instr*.

3.3. Modification du nombre de sorties et fichier d'utilisation

Pour utiliser SIRENE, il faut toujours lui fournir un fichier comprenant les sorties attendues et les entrées correspondantes. Le nombre de valeurs de sortie du fichier doit correspondre au nombre de sorties du réseau utilisé sous peine d'être ignoré.

Lorsque l'apprentissage du réseau est terminé et qu'on veut l'utiliser pour un nouvel ensemble de données, les sorties souhaitées ne sont bien entendu pas disponibles, ni nécessaires. Cependant, le format standard de SIRENE impose de trouver dans le fichier une zone sorties correspondant au nombre de neurones de la dernière couche du réseau. Si ce n'est pas le cas, le deuxième sous-menu permet de faire du "bourrage" dans le fichier à traiter, de manière à obtenir ce format standard. Le troisième sous-menu apporte un degré de liberté supplémentaire. L'utilisateur indique le nom du réseau et du fichier de données à traiter. L'utilisation du programme provoque la conversion, si nécessaire, du fichier de données dans le format adéquat et la création d'un fichier type d'instructions permettant à SIRENE d'utiliser le réseau. Cette fonction a été indispensable pour la méthode de prétraitements NLPCA (cfr. chapitre 5).

3.4. Analyse des résultats

Comme nous l'avons dit précédemment, une fois l'apprentissage réalisé, les ensembles de données sont à nouveau présenté au réseau et SIRENE calcule les résultats pour chaque pattern; c'est-à-dire les sorties obtenues, les erreurs sur ces sorties et le résultat de la classification (correcte ou incorrecte).

Le quatrième sous-programme de ce module interprète ces résultats selon le critère du maximum. Pour chaque pattern, le numéro de la sortie maximale du réseau correspond à la classe obtenue. Le numéro de la sortie maximale des sorties souhaitées du fichier de données correspond à la classe désirée. Le premier résultat du programme est un tableau reprenant pour chaque classe désirée, le nombre de patterns fournis, la classification de SIRENE, c'est-à-dire le nombre de patterns placés dans chaque classe, et le pourcentage de classification correcte. On obtient également le pourcentage global de classification correcte, le pourcentage global de classification impossible (plusieurs sorties maximales égales ou non respect de critères supplémentaires énoncés après) et le pourcentage global de classification incorrecte.

L'utilisateur peut également fournir des critères de décidabilité supplémentaires. Un critère couramment utilisé est la distance minimale. Si les deux plus grandes sorties obtenues du réseau sont trop proches, on considère que le risque de classification incorrecte est trop grand et le pattern est considéré comme inclassifiable. C'est l'utilisateur qui choisit les distances qu'il considère représentatives. Un second critère est la différence entre la sortie obtenue et la sortie souhaitée. Si cette différence est trop grande, on pourrait considérer que le réseau n'a pas reconnu le pattern et n'a donc pas fourni la sortie correspondante ou que le résultat n'est pas réutilisable. Ici encore, c'est l'utilisateur qui indique les différences tolérées. Pour chaque critère supplémentaire, le tableau défini au paragraphe précédent est affiché.

4. Les phases du sommeil

4.1. Description du problème

Une application importante des réseaux de neurones est le traitement des signaux physiologiques de sommeil. Ce problème, plus complexe qu'il n'y paraît de prime abord, ne manque certes pas d'intérêt. En effet, pour une nuit, les enregistrements polygraphiques (électro-encéphalogramme, électro-oculogramme et électro-

myogramme) représentent quelques mille pages et il faut une journée entière à un expert pour en extraire le contenu intéressant et le mettre sous une forme utile. Dans ce contexte, la réalisation d'un analyseur automatique fiable représenterait un gain de temps considérable et une aide réelle au diagnostic pour un certain nombre de maladies. Outre l'aspect éthique d'une telle recherche, il faut noter qu'approximativement une personne sur cent est confrontée au problème de l'apnée du sommeil. On juge donc de l'utilité d'une telle recherche.

Cette recherche a été menée par M. Latour dans le cadre du programme FIRST. Il a montré qu'un réseau de neurones de trois couches permet de classifier un vecteur de données correspondant à une époque d'analyse en une des trois phases du sommeil. Ce réseau fournit des résultats d'un même niveau de performances que d'autres méthodes plus complexes et plus difficiles à mettre en oeuvre.

4.2. Description des données

Dans ce travail, nous avons réutilisé les ensembles de données de M. Latour : trois fichiers de trois mille patterns de données et sorties associées. Une époque du sommeil (un pattern) est défini par cent nombres compris entre -1 et 1. Les sorties associées sont trois nombres : une seule sortie est activée et correspond à la classe. Les deux autres sont nulles. Pour chaque fichier, il y a mille exemples de chaque classe.

4.3. Résultats sans compression

Ne disposant pas des résultats complets de M. Latour, nous avons recommencé la simulation neurale. Cela était de toute manière nécessaire pour pouvoir comparer ultérieurement aux résultats avec compression, qui ont été obtenus par une version plus récente de SIRENE et des conditions d'utilisation différentes. La modification la plus importante est la définition des ensembles de données. Nous disposions, comme dit précédemment, de trois fichiers de taille identique. M. Latour utilisait le premier comme ensemble d'apprentissage et les deux autres comme ensemble test. Dans nos expérimentations, nous avons recomposé les deux premiers, pour obtenir un fichier d'apprentissage de 5 400 patterns (1 800 de chaque classe) et un fichier de validation croisée de 600 patterns (200 de chaque classe). Nous avons gardé le troisième tel quel pour le test. Cette redistribution fournit plus d'exemples au réseau pour l'apprentissage et donc facilite sa généralisation. Les phases du sommeil sont le

seul ensemble de données pour lequel nous n'avons pas utilisé les paramètres de la table 2.1. Nous avons conservé ceux choisis par Monsieur Latour, qui a étudié ce sujet plus en détail. Nous avons dès lors utilisé des fonctions sinus et sigmoïdes pour les neurones, 20 neurones en couche cachée, l'algorithme du gradient conjugué de Polak et Ribière et la fonction somme des carrés des erreurs pour l'apprentissage.

Le premier fichier retourné par SIRENE décrit, pour chaque itération de l'apprentissage, l'évolution du réseau. C'est ce qui est représenté graphiquement à la figure 2.1. Trois courbes sont présentes : une pour la fonction d'apprentissage (échelle de gauche en unité), la fonction de validation (échelle de gauche en unité) et la fonction de test (échelle de droite en pourcentage d'erreur). L'apprentissage s'est arrêté après 2000 itérations. Les trois fonctions passent d'abord par une phase de décroissance rapide. Le réseau corrige ses premières erreurs. Après 1 405 itérations, il a obtenu sa meilleure configuration. Cela est indiqué par le minimum de la courbe de validation. Ensuite, il y a sur-apprentissage et les résultats se dégradent. La fonction d'apprentissage passe de 3 787 à 56. Elle continuera à descendre jusque 25. La fonction de validation débute à 419 et a pour minimum 23. Le pourcentage d'erreur de la fonction test décroît de 65% à 1.7%.

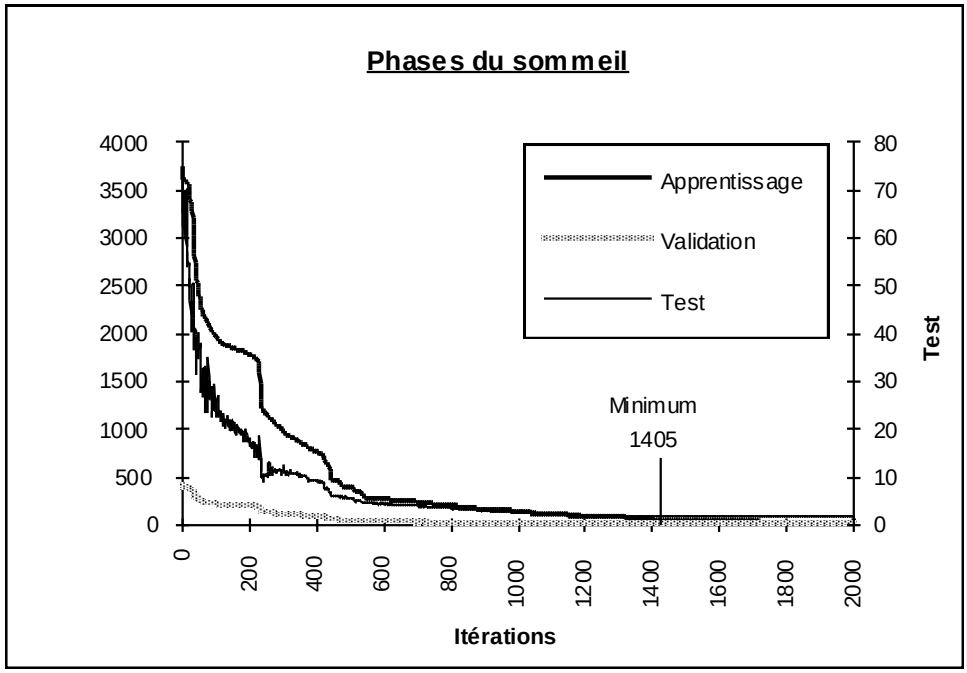


Figure 2.1

Pour réaliser les 2 000 itérations, le programme a travaillé pendant 21 jours. Plus précisément, le temps écoulé était de 484 heures et 25 minutes et le temps CPU de 442 heures et 48 minutes. Nous ne pouvions arrêter l'apprentissage après la 1 405ème itération, car nous ne pouvions savoir s'il s'agissait d'un minimum local ou d'un minimum global.

Commentaire [1]: la différence est le temps pour les I/O et pour les autres prg

Pour le réseau calculé en 1 405 itérations, nous avons obtenu les résultats des tables 2.2 et 2.3. Pour l'ensemble d'apprentissage, 1 789 patterns sur les 1 800 de la classe numéro un ont été classifiés correctement; soit 99.4%. 11 patterns ont été mis à tort dans la classe deux. Il n'y a pas eu d'erreur avec la classe trois. Le principe est le même pour les autres classes attendues. Au total, pour le critère du maximum simple, la classification correcte est de 99.7% pour l'ensemble d'apprentissage et un peu moins pour l'ensemble de test : 98.3%. Cela est normal, vu que le réseau est optimisé par et pour l'ensemble d'apprentissage.

Phases du sommeil - Ensemble d'apprentissage						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	1789	11	0	0	1800	99.4
2	5	1794	1	0	1800	99.7
3	1	0	1799	0	1800	99.9

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	99.7%	99.6%	99.4%
Indécidable :	0%	0.2%	0.3%
Incorrecte :	0.3%	0.2%	0.3%

Table 2.2

Phases du sommeil - Ensemble de test						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	969	31	0	0	1000	96.9
2	0	986	14	0	1000	98.6
3	0	7	993	0	1000	99.3

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	98.3%	98.1%	98.0
Indécidable :	0%	0.4%	0.3%
Incorrecte :	1.7%	1.5%	1.7%

Table 2.3

On constate ici la réelle capacité des réseaux de neurones à s'adapter à ce problème de classification des phases du sommeil. Une fois l'apprentissage réalisé, ces résultats exceptionnels, sont obtenus en moins d'une minute. Il paraît difficile de faire mieux. Il n'est même pas certain que le spécialiste qui aurait passé sa journée à analyser les enregistrements polygraphiques puisse toujours obtenir une aussi bonne classification.

Deux critères supplémentaires ont été utilisés lors de l'analyse des résultats. Le premier impose que la différence entre les deux plus grandes sorties du réseau soit au moins de deux dixièmes. Le second critère réclame que la différence entre la sortie attendue et celle souhaitée soit au maximum de cinq dixièmes. Ces valeurs ont été choisies arbitrairement en tenant compte que les sorties sont comprises entre -1 et 1. La répartition des patterns pour ces deux critères n'étant pas fondamentale, elle n'a pas été décrite par un tableau séparé, mais uniquement par les résultats globaux. Comme on

peut le voir dans les tables 2.2 et 2.3, ces deux critères n'ont pas joué un grand rôle; les résultats sont stables.

5. Les caractères manuscrits de la poste allemande

5.1. Description du problème et des données

Le but recherché ici est la classification d'images de 16 pixels sur 16 comme étant un des chiffres 0 à 9. L'ensemble de données que nous avons reçu comprend 18 000 chiffres digitalisés en niveau de gris (allant de 0 (blanc) à 256 (noir)) au format 16 X 16. Ces chiffres ont été réunis à partir des codes postaux des lettres passant par le service postal de l'ex-République Fédérale d'Allemagne.

Les données fournies sont réparties en deux ensembles de données : apprentissage et test. Chacun des deux ensembles est constitué de 10 fichiers comprenant chacun 900 représentations d'un chiffre. Un vecteur chiffre se compose des 256 pixels de l'image, lue de gauche à droite et de haut en bas.

5.2. Résultats sans compression

La première étape est la création des fichiers de données au format de SIRENE. A chaque caractère, on associe un vecteur de sorties de 10 nombres. Un seul est à un et sa position correspond au chiffre. Les autres sont à zéro. Les fichiers d'apprentissage sont fusionnés de manière à obtenir un fichier SIRENE d'apprentissage de 8 100 patterns (810 pour chaque chiffre) et un fichier SIRENE de validation croisée de 900 patterns (90 pour chaque chiffre). Les fichiers de test sont assemblés, mais deux tiers sont supprimés faute de place sur le disque à notre disposition. Ceci n'a pas de conséquence importante, puisque ce fichier n'est pas utilisé pour l'apprentissage.

Nous n'avons pas réalisé la simulation neurale sans compression. En effet, la taille des données et la taille du réseau qu'elles impliquent posent des problèmes. Expliquons. Définissons d'abord le réseau qui serait nécessaire. Il comprend 256 entrées et 10 sorties. Pour déterminer le nombre de neurones en couche cachées, il faut faire plusieurs essais, jusqu'à l'obtention de bons résultats. Selon la règle des 10%, idéalement nous devrions avoir :

$$(256\,entrées + 10\,sorties) * nbre\_cachés = nbre\_synapses$$
$$nbre\_synapses * 10 < 8100\,patterns$$
$$\Leftrightarrow nbre\_cachés \leq 3$$

Cette règle ne pourra pas être respectée. Trois neurones ne peuvent généraliser 256 données en 10 sorties. Un nombre envisageable de neurones en couche cachée serait par exemple 100. Il y aurait alors 26 600 poids ((256+10)*100) à modifier à chaque itération en fonction de seulement 8 100 exemples. On peut douter de la capacité du réseau à s'adapter. Un tel apprentissage serait long et difficile. Il faudrait de plus le recommencer avec d'autres valeurs que 100 pour améliorer les résultats. Quand on constate le temps qui a été nécessaire à l'apprentissage du réseau, beaucoup plus petit, analysant les phases du sommeil, on comprend que cette démarche n'est pas acceptable. De plus, les données étaient volumineuses et occupaient toute la place disponible sur le disque. Il n'était pas possible de monopoliser un tel espace disque et une machine SUN durant un si grand laps de temps. Notons pour terminer qu'à notre connaissance, personne n'a travaillé sur l'ensemble initial des données. Tout le monde en a fait des compressions par différentes méthodes.

6. Les véhicules

6.1. Description du problème et des données

On cherche à reconnaître une silhouette comme étant celle d'un type de véhicule parmi quatre. On utilise pour cela un ensemble de caractéristiques extraites de la silhouette du véhicule vu sous différents angles. Les quatre types de véhicules sont : un bus à impériale, un van chevrolet, une Saab 9000 et une Opel Manta 400. Il y a 18 attributs réels par véhicule et 846 représentations sont disponibles.

6.2. Résultats sans compression

Cette fois-ci, nous n'avons pas du préparer les données. Nous les avons reçues directement au format de SIRENE. Le réseau choisi pour l'apprentissage utilise les paramètres de la table 2.1. Son architecture comprend 18 neurones en entrée, 5 en couche cachée et 4 pour les sorties. C'est la structure qui a donné les meilleurs résultats et qui était proposée dans la littérature. Cependant, les résultats ne sont pas exceptionnels. Cela est essentiellement du au faible nombre de patterns disponibles

pour l'apprentissage. Pour respecter la règle des dix pourcents, le nombre de neurones en couche cachée aurait du être inférieur ou égale à 3 (18 * 3 + 3 * 4 <= 846 / 10).

Le graphe 2.2 illustre la phase d'étude du réseau. Le meilleur réseau est obtenu après 59 itérations. La fonction d'apprentissage passe de 2 870 à 504. La fonction de validation atteint un pourcentage d'erreurs de 16%, lorsque la fonction de test reste à 17%.

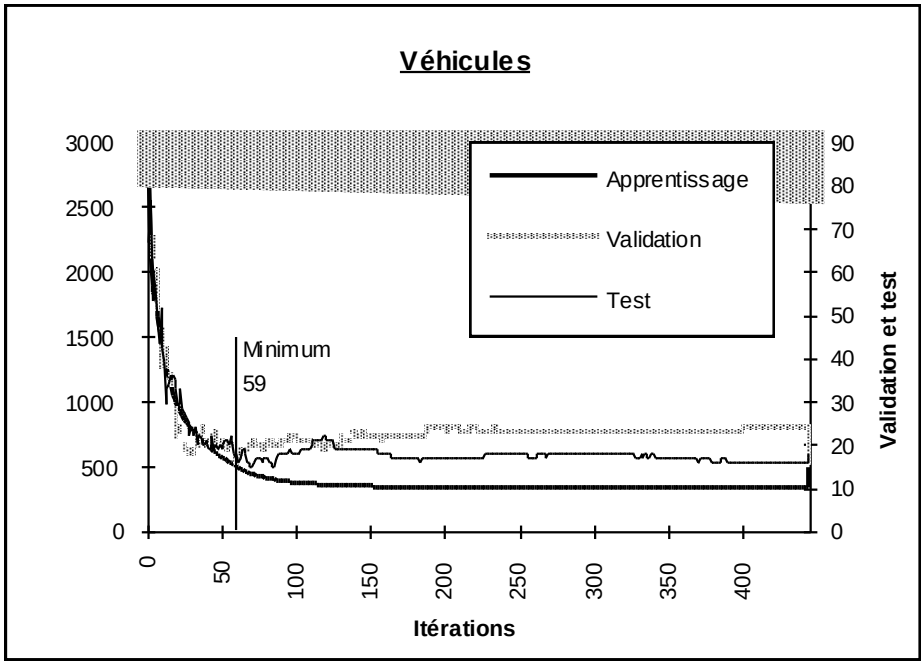


Figure 2.2

1 heure et 29 minutes ont été nécessaires pour réaliser ces 444 itérations. Il y correspond un temps CPU de 1 heure et 22 minutes. Il faut noter ici encore que le meilleur réseau est obtenu bien avant l'arrêt de l'apprentissage. Les meilleurs résultats sont reproduits dans les tables 2.4 et 2.5.

Véhicules - Ensemble d'apprentissage							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	128	37	2	1	0	168	76.1
2	26	136	6	0	0	168	80.9
3	0	2	164	2	0	168	97.6
4	1	2	1	150	0	154	97.4

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	88%	86.6%	74.4%
Indécidable :	0%	2.4%	21.6%
Incorrecte :	12%	11.0%	4.0%

Table 2.4

Véhicules - Ensemble de test							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	18	5	1	0	0	24	75.0
2	5	16	2	1	0	24	66.6
3	0	1	23	0	0	24	95.8
4	0	1	0	23	0	24	95.8

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	83.3%	82.2%	67.8%
Indécidable :	0%	1.4%	28.7%
Incorrecte :	16.7%	16.4%	3.5%

Table 2.5

6.3. Comparaisons avec des résultats officiels

Nous pouvons comparer nos résultats avec ceux du projet ESPRIT StatLog. Son but est l'évaluation des performances de différents algorithmes de classification (statistique, machine intelligente et réseau de neurones) pour des problèmes commerciaux et industriels complexes. Parmi eux, nous trouvons les caractères allemands et les véhicules. La table 2.6 reprend les comparaisons pour le cas des véhicules.

Algorithme	Type	Source	Exactitude (%)		Temps (sec.)	
			Appr.	Test	Appr.	Test
Quadra	Stat.	Strath	91.4	84.9	28	29
Alloc80	Stat.	Leeds	100.0	82.7	30	10

LogReg	Stat.	Strath	83.3	80.9	601	5
Backpropag	Neur.	Strath	83.2	79.3	14 400	4
Discriminant	Stat.	Strath	79.8	78.4	13	5
SMART	Stat.	Leeds	93.8	78.3	2 502	1
C4.5	Mach.	Turing	93.5	73.4	174	2
k-N-N	Stat.	Leeds	100.0	72.5	164	23
CART	Mach.	Granada	76.5	71.6	25	1
CN2	Mach.	Daimler	98.2	70.7	95	1
AC2	Mach.	Isoft	-	70.4	5 525	213
NewID	Mach.	Daimler	99.0	70.3	20	2
INDCART	Mach.	Strath	95.3	70.2	83	1
Radial	Neur.	Strath	90.2	69.3	1 700	12
ITrule	Mach.	Brainwr	-	67.6	985	
Kohonen	Neur.	Luebeck	88.5	66.0	5 692	50
Cal5	Mach.	Fraunhofer	70.1	64.9	39.6	1
Castle	Stat.	Granada	49.5	45.0	3	3
Bayes	Stat.	Strath	47.7	44.2	2	1

Table 2.6

En quatrième position, nous trouvons l'algorithme de rétropropagation. C'est le seul qui utilise un réseau de type Perceptron comme nous. Nous ne disposons malheureusement pas d'informations sur la structure utilisée. Avec le réseau décrit dans le paragraphe précédent, nous obtenons de meilleurs résultats (88.1% et 83.1%), en encore moins de temps (4 920 secondes pour l'apprentissage et 1 seconde pour le test). Nous sommes donc en seconde position. Nous avons réalisé une autre simulation avec un réseau beaucoup plus grand (18-10-4). Dans ce cas, en 27 660 secondes et 1 726 itérations, nous n'obtenions pas de meilleurs résultats : classification correcte à 97.5% pour l'apprentissage, mais 80.4% pour le test. On constate que le réseau a très bien appris son ensemble d'exemples. Par contre, il n'a pas réussi à les généraliser aussi bien à d'autres présentations. Cela confirme que le nombre d'exemples d'apprentissage était trop faible pour ce réseau (658 exemples pour 220 synapses).

7. Données corrélées

Il ne s'agit pas ici d'un problème de classification. Cet ensemble de données a été créé de toutes pièces pour illustrer la méthode de compression NLPCA. Nous attendrons donc ce chapitre pour le présenter.

8. Conclusions

Nous avons présenté les ressources que nous utiliserons tout au long de ce travail : les données, les programmes et les ordinateurs. La séquence d'opérations est toujours la même : utilisation de notre programme pour réaliser les prétraitements, utilisation de SIRENE pour simuler un réseau de neurones de reconnaissance de formes et analyse des résultats. Pour faciliter ces manipulations, un module d'outils a été écrit.

Nous avons également décrit la classification sans prétraitement des ensembles de données à notre disposition. Les réseaux de neurones ont montré leur capacité à s'adapter aux problèmes posés. Cependant, les temps d'apprentissage qui ont été nécessaires sont énormes. Pour le problème de la reconnaissance des caractères manuscrits, nous n'avons même pas pu réaliser une simulation. Des méthodes doivent absolument être développées pour pouvoir utiliser d'importants ensembles de données dans des réseaux de neurones. Les chapitres suivants décrivent de tels procédés.

CHAPITRE 3
LA MÉTHODE DE KARHUNEN-LOÈVE

1. Introduction

La méthode de Karhunen-Loève est une technique pour représenter un échantillon d'une fonction générée par un processus aléatoire. Il a été montré qu'elle minimisait l'erreur au sens des moindres carrés. La méthode de Karhunen-Loève extrait donc un ensemble de caractéristiques qui est optimal pour représenter un pattern dont la source est aléatoire.

Ce chapitre est constitué de trois parties. La première décrit la méthode de Karhunen-Loève. La seconde présente notre implémentation. Enfin, la troisième comprend les principaux résultats de son application sur nos ensembles de données.

2. La méthode

2.1. Description formelle

La méthode de Karhunen-Loève est une méthode d'approximation d'un ensemble de fonctions continues de temps par un développement en série. Soit l'ensemble de fonctions $f_i(t) \quad (i = 1,2,\ldots,N)$, le résultat est une combinaison linéaire de fonctions de base, $\phi_j(t) \quad (j = 1,2,\ldots)$ de la forme :

$$f_i(t) = \sum_{j=1}^{\infty} \alpha_{ij} \phi_j(t) \tag{1}$$

Les fonctions de base sont obtenues par résolution de l'équation suivante :

$$\lambda_j \phi_j(t) = \int_{-\infty}^{+\infty} R(t,\tau)\, \phi_j(\tau)\, d\tau \tag{2}$$

où $R(t,\tau)$ est la fonction d'autocorrélation des f(t) et est donnée par :

$$R(t,\tau) = \underset{i}{E}[f_i(t) f_i(\tau)], \tag{3}$$

où E[.] est la moyenne sur les N fonctions de l'ensemble. On obtient ensuite les coefficients α_{ij} par :

$$\alpha_{ij} = \int_{-\infty}^{+\infty} f_i(t)\phi_j(t)\,dt$$

Dans notre cas, nous voulons transformer un vecteur de données (un pattern) en un autre de taille plus petite. Les éléments d'un vecteur peuvent être vus comme un échantillonnage d'une fonction continue. Le nombre de points d'échantillon, n, étant choisi de manière à retenir suffisamment d'informations. Nous devons donc adapter les définitions précédentes au cas discret. Nos fonctions peuvent alors être représentées par les vecteurs colonnes :

$$F_i = \begin{bmatrix} f_i(t_1) \\ f_i(t_2) \\ \vdots \\ f_i(t_n) \end{bmatrix} \quad (i = 1, 2, \ldots, N) \tag{4}$$

L'équation (1) devient alors la somme finie

$$F_i = \sum_{j=1}^{n} \alpha_{ij}\, \Phi_j \tag{5}$$

où Φ_j est la représentation vectorielle de la j^{ème} fonction de base :

$$\Phi_j = \begin{bmatrix} \phi_j(t_1) \\ \phi_j(t_2) \\ \vdots \\ \phi_j(t_n) \end{bmatrix} \quad (j = 1, 2, \ldots, n) \tag{6}$$

De même, (2) et (3) sont remplacés par

$$\lambda_j\, \Phi_j = S\, \Phi_j \tag{7}$$

et

$$S = \underset{i}{E}[F_i F_i^T], \tag{8}$$

Les équations (7) et (8) montrent que S est une matrice d'autocorrélation et que λ_j et Φ_j sont les $j^{ème}$ valeur propre et vecteur propre de S. Puisque les fonctions de base sont des vecteurs propres, elles sont orthogonales; c'est-à-dire :

$$\Phi_j^T \Phi_i = \begin{cases} 1 & pour\ i = j \\ 0 & pour\ i \neq j \end{cases} \tag{9}$$

Dès lors, les coefficients α_{ij} de l'expression peuvent être obtenus par :

$$\alpha_{ij} = \Phi_j^T\ F_i \tag{10}$$

ou, en notation matricielle,

$$A_i = B^T F_i$$

avec

$$A_i = \begin{bmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \vdots \\ \alpha_{in} \end{bmatrix} \quad et \quad B = \left[\Phi_1 \Phi_2 \cdots \Phi_n\right]$$

Quand les n fonctions de base sont utilisées, les F_i sont obtenus sans erreur. Par contre, si nous en sélectionnons moins, l'expansion de Karhunen-Loève devient une approximation. L'erreur au sens des moindres carrés est calculée directement par :

$$\varepsilon^2 = \underset{i}{E}\left[\left(F_i - \sum_{j=1}^{k} \alpha_{ij}\Phi_j\right)^T \left(F_i - \sum_{j=1}^{k} \alpha_{ij}\Phi_j\right)\right] \tag{11}$$

Par (9) et (10), on peut simplifier :

$$\varepsilon^2 = \sum_{j=k+1}^{n} \Phi_j^T \underset{i}{E}[F_i F_i^T]\Phi_j\,, \tag{12}$$

et on obtient par (8) que

$$\varepsilon^2 = \sum_{j=k+1}^{n} \lambda_j \tag{13}$$

Si les vecteurs propres sont rangés par ordre de valeur propre décroissante, on minimise ainsi l'erreur moyenne au sens des moindres carrés. On voit de plus, que plus le nombre de vecteurs propres conservés est grand, moins l'erreur sera grande et même nulle si il n'y a pas de réduction.

La matrice A ainsi obtenue est l'ensemble des nouveaux patterns d'entrées pour le réseau de neurones. Notons que la matrice B n'est utilisée que pour le calcul de A et n'est pas conservée. A étant une corrélation de B et des patterns originaux, elle devrait suffire au réseau de neurones. Cependant, l'erreur calculée précédemment par les formules (11), (12) et (13) ne peut plus être celle liée à l'utilisation du réseau étant donné que ces formules utilisent la matrice B. L'erreur sera mesurée par comparaisons avec les résultats obtenus sans prétraitement.

2.2. Signification géométrique

Dans cette méthode, on part d'un tableau de données F où chacune des N lignes correspond à un individu et chacune des n colonnes correspond à une caractéristique.

La première opération réalisée est le recentrage du tableau. Pour chaque caractéristique j, la moyenne m_j des N individus est calculée :

$$m_j = \frac{1}{N} \sum_{i=1}^{N} F_{ij}$$

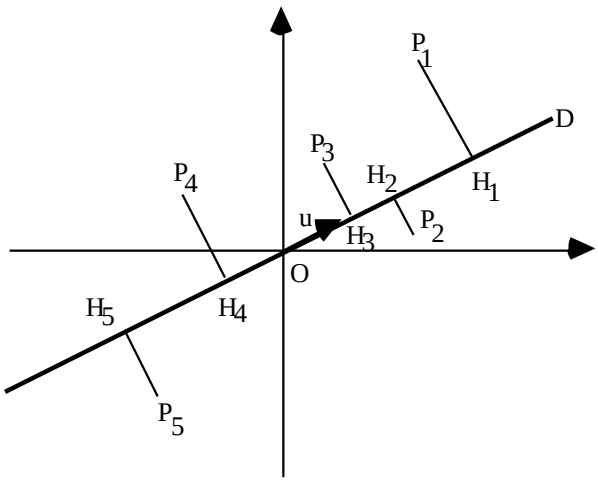
On obtient alors le tableau recentré par l'opération suivante :

$$\overline{F}_{ij} = F_{ij} - m_j$$

A chaque ligne i de $\overline{F}$, on peut associer le point P_i de coordonnées $\overline{F}_{ij}$ pour $1 \leq j \leq n$. Le résultat est un nuage de N points.

Le but de la méthode est de représenter le plus simplement possible les corrélations entre les n caractéristiques. Pour cela, soit D une droite dans IR^n de direction $\vec{u}$ avec $\|\vec{u}\| = 1$. Soit H_i la projection orthogonale de P_i sur D. On cherche u qui minimise :

$$\sum_{i=1}^{N} \left| \vec{H_i P_i} \right|^2$$



On a :

$$\vec{OH_i} = (\vec{OP_i} | \vec{u})\, \vec{u},$$

où

$$(\vec{OP_i} | \vec{u}) = \sum_{j=1}^{n} \overline{F}_{ij} u_j ,$$

donc :

$$\sum_{i=1}^{N} \left| \vec{OH_i} \right|^2 = \sum_{i=1}^{N} (\vec{OP_i} | \vec{u})^2$$

Les P_i étant donnés, chercher $\vec{u}$ qui minimise $\sum_{i=1}^{N}\left|\vec{H_iP_i}\right|^2$ est équivalent, d'après le théorème de Pythagore, à chercher $\vec{u}$ qui maximise $\sum_{i=1}^{N}\left|\vec{OH_i}\right|^2$. D'après un théorème de Courant-Fischer, cela revient à chercher le vecteur propre $\vec{u}_1$ correspondant à la plus grande valeur propre λ_1 de $\overline{F}^{\mathrm{T}}\overline{F}$. En déterminant les vecteurs propres correspondant aux valeurs propres suivantes, on obtient une représentation plus complète.

3. Le programme

Tous les programmes ont été écrits à l'aide du langage C. Le code de la méthode de Karhunen est fourni en annexe.

L'implémentation est tirée directement de la théorie. L'exécution comprend trois étapes : la recherche des vecteurs propres, la détermination des vecteurs qui appartiendront à la matrice de transformation et la phase de compression proprement dite.

```
+++ KARHUNEN - LOEVE +++

  0. Aide

  1. Vecteurs propres
  2. Matrice de transformation
  3. Creation fichier compresse

  9. Retour au menu principal
```

3.1. Vecteurs propres

L'utilisateur a le choix entre calculer la matrice des vecteurs propres d'un nouvel ensemble de données, enregistrer une nouvelle matrice ou charger une ancienne.

3.2. Matrice de transformation

L'utilisateur a les mêmes possibilités : création, enregistrement ou chargement. Pour la création d'une nouvelle matrice de transformation, trois critères de sélection du nombre de vecteurs propres à utiliser sont possibles : constant, selon les valeurs propres, selon les valeurs propres avec limite maximale.

En utilisant chaque fois la matrice de transformation de l'ensemble d'apprentissage pour les deux autres ensembles de données, on augmente la corrélation entre l'apprentissage et l'utilisation du réseau. Cela améliore les résultats.

3.3. Création du fichier compressé

L'utilisateur peut décider en plus de calculer l'erreur moyenne (définie comme précédemment) pour chaque pattern ou globalement. Il peut également demander que les sorties soient comprises dans l'intervalle [-1,1]; ce qui est très pratique lors de l'utilisation d'un réseau de neurones dont les fonctions habituelles imposent cette condition. Cette borne est obtenue par division des vecteurs propres par une constante adéquate.

4. Analyse des phases du sommeil

4.1. Introduction

Les données du sommeil, présentées dans le chapitre 2, sont le premier ensemble qui a été prétraité par la méthode de Karhunen. Nous ne pouvions prédire le nombre de vecteurs propres à conserver, c'est-à-dire l'importance de la compression, pour conserver une classification valable. Nous avons donc procédé par essais. Nous présentons ici les deux résultats les plus significatifs : une compression de 100 données à 20 et une de 100 à 10.

4.2. Compression à 20

L'ensemble de données initial a été prétraité par Karhunen pour lui donner un taille des entrées de 20 au lieu de 100. A partir de ces données comprimées, un nouveau réseau a du réaliser la classification en phases du sommeil. L'architecture choisie emploie les mêmes paramètres que le réseau utilisé sans compression préalable. Le réseau comprend 20 neurones en entrée, 9 en couche cachée et 3 en sortie. Ce choix respecte largement la règle des dix pourcents (20 * 3 + 3 * 9 << 8 100 / 10) et a donné de bons résultats.

La figure 3.1 décrit l'apprentissage de ce réseau. Les premières erreurs sont rapidement corrigées. La fonction de validation démarre à la valeur 595 pour atteindre très vite, après seulement 651 itérations, son minimum à la valeur 117. A ce moment, la fonction d'apprentissage a diminué de 5 345 à 731 et la fonction de test est passée de 67% d'erreurs à 11%. Après la 651ème itération, les résultats se dégradent pour l'ensemble de test qui atteint 16% d'erreurs à la fin de la simulation.

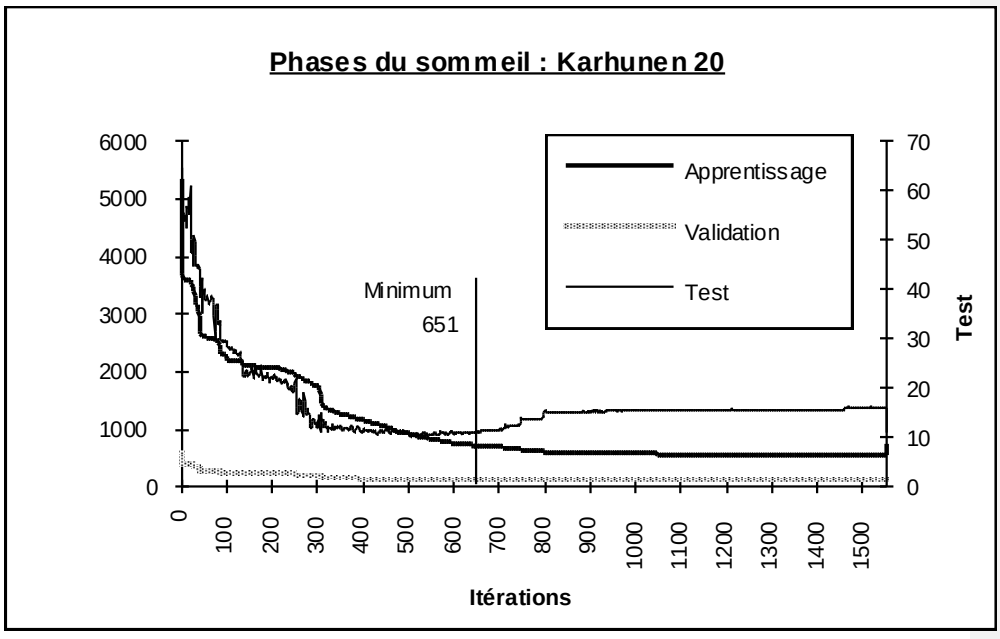


Figure 3.1

Pour réaliser les 1 550 itérations de cette simulation, un ordinateur a été occupé pendant cinq jours : le temps réel est de 109 heures et 31 minutes, mais le temps

CPU n'est que de 54 heures et 4 minutes. Rappelons cependant que les derniers résultats sauvés par SIRENE l'ont été bien avant cela. Ils étaient déjà utilisables à partir de l'itération 651.

Pour le réseau optimum, les résultats détaillés sont reproduits dans les tables 3.1 et 3.2. La simulation permet de reconnaître relativement bien et avec une même certitude chaque classe de l'ensemble d'apprentissage. 91.2% de classifications correctes est un résultat honorable. On constate de plus, et avec plaisir, que le réseau arrive à généraliser sa classification à d'autres exemples qu'il ne connaissait pas. Dans ce cas, il obtient toujours une classification correcte à 89.5%.

Phases du sommeil Karhunen 20 - Ensemble d'apprentissage						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	1587	211	2	0	1800	88.2
2	16	1714	70	0	1800	95.2
3	9	168	1623	0	1800	90.2

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	91.2%	90.1%	90.5%
Indécidable :	0%	2.6%	1.5%
Incorrecte :	8.8%	7.3%	8.0%

Table 3.1

Phases du sommeil Karhunen 20 - Ensemble de test						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	820	180	0	0	1000	82.0
2	0	963	37	0	1000	96.3
3	8	91	901	0	1000	90.1

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	89.5%	88.7%	89.0%
Indécidable :	0%	2.2%	0.8%
Incorrecte :	10.5%	9.1%	10.2%

Table 3.2

Ici encore, la classification est aisée. Les deux critères de décision supplémentaires ne modifient pratiquement pas les résultats : les sorties sont nettes.

4.3. Compression à 10

Une compression à 20 donnant toujours des résultats acceptables, nous avons poussé la compression jusqu'à 10 pour en fixer les limites. Cela permettra de plus des comparaisons avec d'autres méthodes de compression qui supportent une telle diminution du nombre des caractéristiques.

Les données prétraitées ont été présentées à un nouveau réseau définit similairement à ceux des autres analyses des phases du sommeil. Seule sa structure change; il comprend 10 neurones en entrée, 5 en couche cachée et 3 en sortie.

La figure 3.2 représente son apprentissage. Les résultats sont nettement moins bons. La fonction d'apprentissage part de 4 976 pour atteindre sa valeur finale de 2 088. Le réseau n'arrive plus à obtenir de meilleurs résultats et arrête son étude. La fonction de validation, débutant de 552, permettra de sauver juste avant le meilleur réseau qu'elle a trouvé. Son minimum se situe à l'itération 486 avec une valeur de 236. Le pourcentage d'erreurs pour l'ensemble de test s'est stabilisé à 27%.

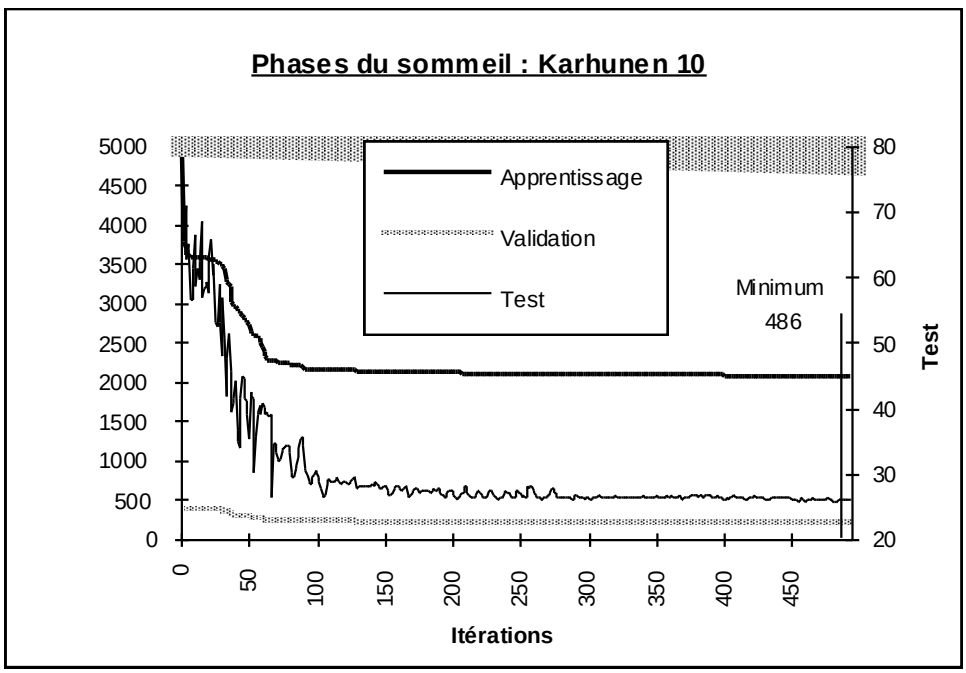


Figure 3.2

Ces résultats ont été obtenus beaucoup plus vite. En temps réel, ces 492 itérations ont nécessité 30 heures et 23 minutes. En temps CPU, cela a demandé 8 heures et 10 minutes. Notons ici que même si le réseau obtenu a été sauvé à l'itération 486, un réseau sauvé dès l'itération 150 aurait donné des résultats très proches : les courbes sont régulières.

Il est intéressant de regarder la classification détaillée du réseau. Elle est présentée dans les tables 3.3 et 3.4. On constate que le réseau généralise parfaitement son apprentissage au fichier test. Les résultats sont identiques : classification correcte d'environ 71% dans les deux cas. On découvre également la source d'erreurs. L'apprentissage a échoué pour la classe deux, alors qu'il est valable pour les autres catégories. Cet échec se propage dès lors à l'ensemble test.

Phases du sommeil **Karhunen 10 - Ensemble d'apprentissage**						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	1551	235	14	0	1800	86.2
2	1027	652	121	0	1800	36.2
3	50	130	1620	0	1800	90.0

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	70.8%	58.2%	53.8%
Indécidable :	0%	20.0%	29.7%
Incorrecte :	29.2%	21.8%	16.5%

Table 3.3

Phases du sommeil **Karhunen 10 - Ensemble de test**						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	918	82	0	0	1000	91.8
2	504	379	117	0	1000	37.9
3	40	70	890	0	1000	89.0

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	72.9%	59.7%	56.5%
Indécidable :	0%	19.6%	27.5%
Incorrecte :	27.1%	20.7%	16.0%

Table 3.4

Une compression jusqu'à dix par la méthode de Karhunen n'est donc pas acceptable. Le résultat est une incapacité du réseau à reconnaître la classe deux, qu'il confond essentiellement avec la classe un. Une compression jusqu'à vingt entrées apparaît comme la limite.

5. Reconnaissance des caractères manuscrits

5.1. Introduction

Pour ce problème, nous avons d'abord du décider de la composition des réseaux à essayer. Avec 8 100 patterns d'apprentissage, cette architecture devait de préférence être limitée à 810 synapses; ce qui est peu pour un problème original de 256 entrées. Dans la littérature, nous avons trouvé des tentatives de classification de ce problème à partir de données réduites à la taille de 40. Nous avons donc d'abord testé cette valeur.

Cela n'a pas abouti. En effet, nos données originales étaient en nuances de gris (valeurs de 0 à 255). Cependant, notre fonction tangente hyperbolique n'est vraiment efficace qu'autour de l'origine. Plus on s'en éloigne, moins la fonction varie. Nous avons donc retraité nos données pour les recentrer dans l'intervalle [-1,1]. Pour cela, nous utilisions une fonction de conversion de SIRENE. Les résultats ont directement suivi.

Nous ne nous sommes pas limité à une compression à 40. Nous avons essayé d'aller plus loin, jusque 20. Nous présentons maintenant en détail ces deux simulations. Nous montrerons ensuite brièvement la différence pour d'autres valeurs recentrées ou non et nous terminerons par une comparaison avec les résultats de la littérature : le projet StatLog.

5.2. Compression à 40 et recentrage

Comme annoncé, nous avons d'abord réduit nos ensembles de données par la méthode de Karhunen et nous les avons ensuite recentrés, grâce à SIRENE, dans l'intervalle [-1,1]. Il faut signaler qu'il a été obligatoire d'utiliser la même matrice de transformation de Karhunen et de centrage pour les trois fichiers. Dans le cas contraire, l'apprentissage s'arrête après quelques itérations et avec un pourcentage d'erreurs énorme.

Le réseau créé pour vérifier la capacité de classification après compression, comprend 40 neurones en entrée et en sortie, 1 pour chaque classe. L'importance de la couche cachée a été décidée selon le critère des dix pourcents. Le maximum est dès lors de 16 neurones (40 * 16 + 16 * 10 = 800 < 8 100 / 10). Étant donné les excellents résultats obtenus, il n'a pas été nécessaire d'essayer d'autres valeurs.

Lors de notre simulation, nous n'avions pas assez d'espace disque pour utiliser l'ensemble test de 9 000 patterns au complet. Nous en avons extrait un tiers. Cependant, pour permettre des comparaisons avec le projet StatLog qui emploie le fichier entier, nous avons après calculé les résultats détaillés pour les 9 000 patterns de test.

L'évolution de l'apprentissage est expliquée par la figure 3.3. Cela a été très rapide. En 53 itérations, la fonction de validation a atteint le minimum de 8% d'erreurs. Pour les 3 000 exemples, la fonction de test indique à ce moment une erreur de 7%. La fonction d'apprentissage est partie de 87 796 pour tomber à 2 046. Après la 53ème itération, les erreurs recommencent à augmenter progressivement. Après 580 itérations, les résultats ne pouvant plus s'améliorer, nous avons décidé d'arrêter nous-mêmes l'apprentissage et de libérer l'ordinateur.

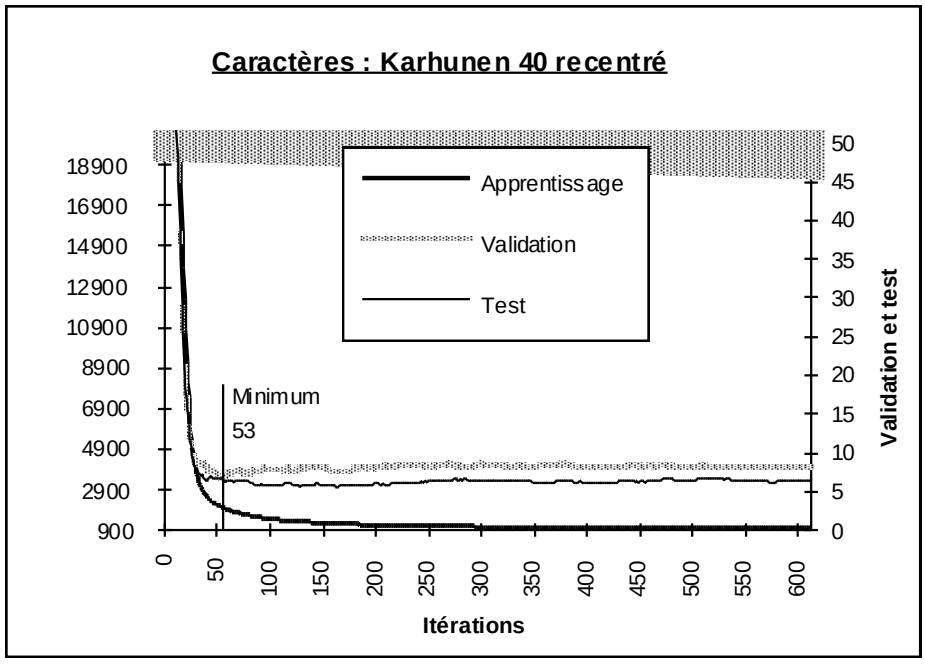


Figure 3.3

Une dizaine d'heures ont été nécessaires pour obtenir le meilleur réseau. Nous avons arrêté l'apprentissage à la 580ème itération après 87 heures et 20 minutes réelles et 86 heures et 31 minutes CPU.

La table 3.5 détaille les résultats de la classification de l'ensemble d'apprentissage. Les pourcentages spécifiques et globaux de classification sont tous similaires et très élevés. Les erreurs sont équitablement réparties dans chaque classe, il ne s'agit pas d'un problème de classification de caractères se ressemblant, mais d'une limite du réseau. Avec 96.3% d'exactitude, on peut parler de réussite.

Caractères - Karhunen 40 recentré - Ensemble d'apprentissage

Classe	Classe obtenue											∑	%
	1	2	3	4	5	6	7	8	9	10	?		
1	769	4	2	0	4	1	2	3	17	8	0	810	94.9
2	10	792	0	0	0	0	0	1	2	5	0	810	97.7
3	0	0	788	5	0	3	1	1	5	7	0	810	97.3
4	0	0	9	769	0	7	3	1	9	12	0	810	94.9
5	3	1	0	0	791	1	10	1	2	1	0	810	97.6
6	1	0	0	13	1	765	6	2	8	14	0	810	94.4
7	1	0	0	1	3	6	795	0	4	0	0	810	98.1
8	8	2	1	1	6	2	0	785	2	3	0	810	96.9
9	1	3	4	1	0	6	9	2	768	16	0	810	94.8
10	4	0	4	5	7	4	1	2	5	778	0	810	96.0

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	96.3%	95.9%	93.4%
Indécidable :	0%	1.1%	5.2%
Incorrecte :	3.7%	3%	1.4%

Caractères - Karhunen 40 recentré - Ensemble de test

Classe	Classe obtenue											∑	%
	1	2	3	4	5	6	7	8	9	10	?		
1	824	24	1	0	7	0	10	4	25	5	0	900	91.6
2	9	871	0	3	0	1	0	7	3	6	0	900	96.8
3	2	3	824	12	0	16	6	4	19	14	0	900	91.6
4	0	0	24	814	0	15	0	10	15	22	0	900	90.4
5	14	4	2	0	845	2	9	6	11	7	0	900	93.9
6	5	0	9	24	0	815	17	4	9	17	0	900	90.6
7	6	0	1	1	9	10	869	0	2	2	0	900	96.6
8	6	7	3	0	6	2	0	854	15	7	0	900	94.9
9	5	1	11	3	0	15	13	4	825	23	0	900	91.7
10	4	8	3	12	8	11	0	14	14	826	0	900	91.8

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	93%	92.1%	89.9%

Indécidable :	0%	2.3%	6.7%
Incorrecte :	7%	5.6%	3.4%

Table 3.5 et 3.6

L'ensemble complet de test (9 000 patterns) a été utilisé pour vérifier l'exactitude de la classification. La table 3.6 montre ces résultats. Ils sont semblables à ceux de l'ensemble réduit testé lors de l'apprentissage. Le pourcentage de classification correcte est de 93%. Les résultats sont précis. Les deux critères supplémentaires ne modifient pas énormément les pourcentages. Notons qu'en utilisant ces critères, on peut diminuer le taux d'erreur en augmentant le taux d'incertitude. Les éléments non classifiables peuvent être donnés à une autre procédure de décision.

5.3. Compression à 20 et recentrage

Les mêmes opérations ont été réalisées pour obtenir 20 valeurs entre -1 et 1 pour chaque pattern de données. Ces valeurs ont été présentées à un réseau de 20 neurones en entrée et 10 en sortie. Nous avons décidé d'utiliser 16 neurones en couche cachée comme pour le réseau précédent. Le nombre d'exemples disponibles nous le permet et cela facilitera la généralisation.

La figure 3.4 décrit l'apprentissage. Les courbes de validation et de test se superposent et décroissent durant une longue période. La fonction d'apprentissage suit la même direction. Elle part de 94 000 pour atteindre

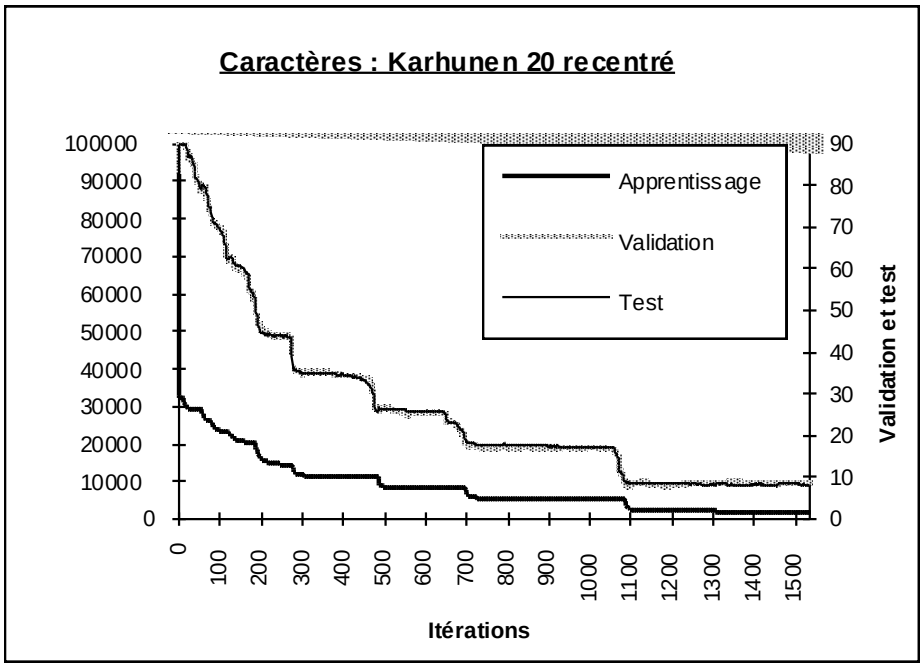


Figure 3.4

Cet apprentissage a été beaucoup plus long : 1 097 itérations ont été nécessaires pour obtenir le meilleur réseau. Nous avons arrêté volontairement l'apprentissage après 1 500 itérations, soit 291 heures et 42 minutes réelles (12 jours) ou 285 heures et 44 minutes de CPU. Le taux de classification aurait peut-être pu encore diminuer de quelques dixièmes de pourcent, mais le temps d'apprentissage, excessif, suffit à déclarer cette taille de compression comme inadaptée. Nous avons donc libéré l'ordinateur de cette charge de travail.

Les tables 3.7 et 3.8 reprennent les résultats. Ici aussi, ils sont excellents. La méthode de Karhunen-Loève permet des compressions très importantes. Pour une compression à 20, le pourcentage de classification correcte de l'ensemble de test est de 91%. Lorsque le réseau se trompe, la classe deux est plus souvent choisie.

Caractères - Karhunen 20 recentré - Ensemble d'apprentissage													
Classe	Classe obtenue											Σ	%
	1	2	3	4	5	6	7	8	9	10	?		
1	775	17	2	0	7	0	2	1	3	3	0	810	95.7
2	6	788	1	0	10	0	0	5	0	0	0	810	97.3
3	2	22	754	1	2	2	9	3	7	2	6	810	93.1

4	0	10	5	777	1	1	0	2	1	12	1	810	95.9
5	2	17	0	0	775	0	5	2	0	9	0	810	95.7
6	1	5	0	1	2	791	4	2	0	4	0	810	97.7
7	1	7	1	0	7	8	783	0	3	0	0	810	96.7
8	3	17	4	2	7	0	0	774	0	3	0	810	95.6
9	2	13	6	3	1	6	7	1	759	8	4	810	93.7
10	2	24	0	9	14	1	1	4	8	742	5	810	91.6

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	95.3%	94.8%	93.1%
Indécidable :	0.2%	2.6%	5.0%
Incorrecte :	4.5%	2.6%	1.9%

Table 3.7

Bien que ces résultats soient excellents, cette taille de compression ne peut nous satisfaire. En effet, notre objectif est de réduire la dimension des vecteurs pour accélérer l'apprentissage des réseaux de neurones. Une compression plus importante n'implique pas forcément un temps d'apprentissage plus court. C'est le cas ici, la taille de compression de 40, bien que moins importante, permet un apprentissage plus rapide.

Caractères - Karhunen 20 recentré - Ensemble de test													
Classe	Classe obtenue											∑	%
	1	2	3	4	5	6	7	8	9	10	?		
1	269	15	0	0	6	0	0	1	7	0	2	300	89.7
2	2	292	0	0	1	0	0	2	1	2	0	300	97.3
3	0	8	271	4	1	1	1	3	2	1	8	300	90.3
4	0	6	4	276	0	5	0	0	0	5	4	300	92.0
5	2	10	0	0	271	2	5	1	0	9	0	300	90.3
6	0	8	0	2	2	283	2	0	1	1	1	300	94.3
7	2	3	1	0	4	3	286	0	1	0	0	300	95.3
8	3	8	1	1	3	1	0	277	1	3	2	300	92.3
9	1	14	5	4	1	2	7	1	260	2	3	300	86.7
10	2	20	0	7	6	3	2	1	3	254	2	300	84.7

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	91.3%	89.8%	88.9%
Indécidable :	0.7%	5.4%	7.1%
Incorrecte :	8.0%	4.8%	4.0%

Table 3.8

5.4. Autres tentatives de compression

Nous présentons ici les résultats globaux obtenus pour d'autres tailles de compression. La table 3.9 reprend pour chacune, le numéro de la meilleure itération, le pourcentage de classification correcte pour les deux ensembles de données à ce moment, le nombre d'itérations avant l'arrêt de l'apprentissage et les différents temps qui correspondent à la simulation entière. Ces dernières informations doivent être interprétées pour estimer le temps nécessaire à l'obtention des meilleurs résultats. Pour cela, il suffit de savoir que chaque étape nécessite une même durée. Le temps estimatif pour les données comprimées à 40 et recentrées est donc d'environ 8 heures.

Méthode	Itération	Exactitude apprentis.	Exactitude test	Nombre itérations	Temps réel	Temps CPU
40 recentré	53	95.0%	93.0%	580	87h 20'	86h 31'
30 recentré	2 861	98.1%	94.4%	3 000	400h 44'	394h 08'
20 recentré	1 097	95.3%	91.3%	1 498	291h 42'	285h 44'
40	213	46.0%	45.0%	249	40h 21'	38h 09'
30	131	40.0%	38.0%	134	29h 27'	18h 12'

Table 3.9

La compression par la méthode de Karhunen à une taille de 40, suivie d'un recentrage par SIRENE, donne d'excellents résultats et les meilleurs pour nos essais. Les autres tailles recentrées donnent aussi de bons résultats, mais en beaucoup plus de temps; ce qui va à l'encontre de notre objectif.

5.5. Comparaisons avec des résultats officiels

Dans le cadre du projet ESPRIT StatLog, le problème de classification des caractères manuscrits a été traité. Leurs données ont été préalablement compressées par la méthode de Karhunen. Les caractéristiques ont été ramenées à une taille de 40. Ces données ont alors été présentées aux 19 algorithmes de la table 3.10.

Algorithme	Type	Source	Exactitude (%)		Temps (sec.)	
			Appr.	Test	Appr.	Test
k-N-N	Stat.	Leeds	100.0	98.0	6 706	
Quadra	Stat.	Strath	98.7	97.9	930	863

Alloc80	Stat.	Leeds	100.0	97.6	-	23 279
Backpropag	Neur.	Strath	95.9	95.1	129600	2 400
Radial	Neur.	Strath	95.2	94.5	1 700	580
SMART	Stat.	Leeds	95.7	94.3	174965	58
Discriminant	Stat.	Strath	93.0	92.5	87	54
Castle	Stat.	Granada	87.4	86.5	4 535	56 053
NewID	Mach.	Daimler	100.0	83.8	779	109
AC2	Mach.	Isoft	100.0	83.2	15 155	14 086
INDCART	Mach.	Strath	99.7	83.0	3508	47
CN2	Mach.	Daimler	96.4	82.0	2 902	100
C4.5	Mach.	Turing	95.0	82.0	1 437	35
Bayes	Stat.	Strath	79.5	77.7	65	76
Cal5	Mach.	Fraunhofer	75.2	66.9	3 053	64
LogReg	Stat.	Strath	Échec	Échec	Échec	Échec
ITrule	Mach.	Brainwr	-	-	-	-
Cart	Mach.	Granada	-	-	-	-
Kohonen	Neur.	Luebeck	-	-	-	-

Pour les trois derniers algorithmes, les résultats n'ont pas été communiqués.

Table 3.10

Nos résultats sont légèrement inférieurs à l'algorithme de la rétropropagation, mais restent fort proches. Nous nous classons en 6ème position sur les 20 méthodes. Les résultats étant proches, nous pouvons considérer notre méthode comme compétitive.

6. Reconnaissance des véhicules

6.1. Introduction

Le dernier problème que nous avons traité par Karhunen est celui de la reconnaissance des silhouettes de véhicules. En réalité, avec 18 entrées, ce problème n'a pas besoin d'être réduit pour être traité par un réseau de neurones. Nous ne l'avons fait que dans un but de comparaison avec la méthode de prétraitement NLPCA que nous verrons plus loin. Trois tailles de compression ont été tentées : 10, 8 et 4. Nous allons détailler les deux premières : la compression à 10 car elle donne les meilleurs résultats et la compression à 8 car nous la comparerons ultérieurement aux résultats de la méthode NLPCA. La compression à 4 sera résumée. Rappelons que nous possédions des résultats de références et qu'ils sont reproduits dans le chapitre 2.

6.2. Compression à 10.

Les données réduites de la taille 18 à 10 par Karhunen, ont été présentées à un réseau de 10 entrées, 4 sorties et 8 neurones en couche cachée. Son apprentissage est décrit à la figure 3.4. La fonction d'apprentissage part de 2 941 pour atteindre 590 à l'itération 61. Cette itération correspond au minimum de la fonction d'apprentissage : 17% d'erreurs. A ce moment, la fonction test équivaut à 24% d'erreurs. Comme on peut le voir sur le graphique, ces valeurs restent stationnaires après. Il était inutile de poursuivre la simulation jusqu'à l'itération 975. Le temps réel d'utilisation de l'ordinateur a été de 3 heures et 9 minutes et le temps CPU a été de 2 heures et 57 minutes.

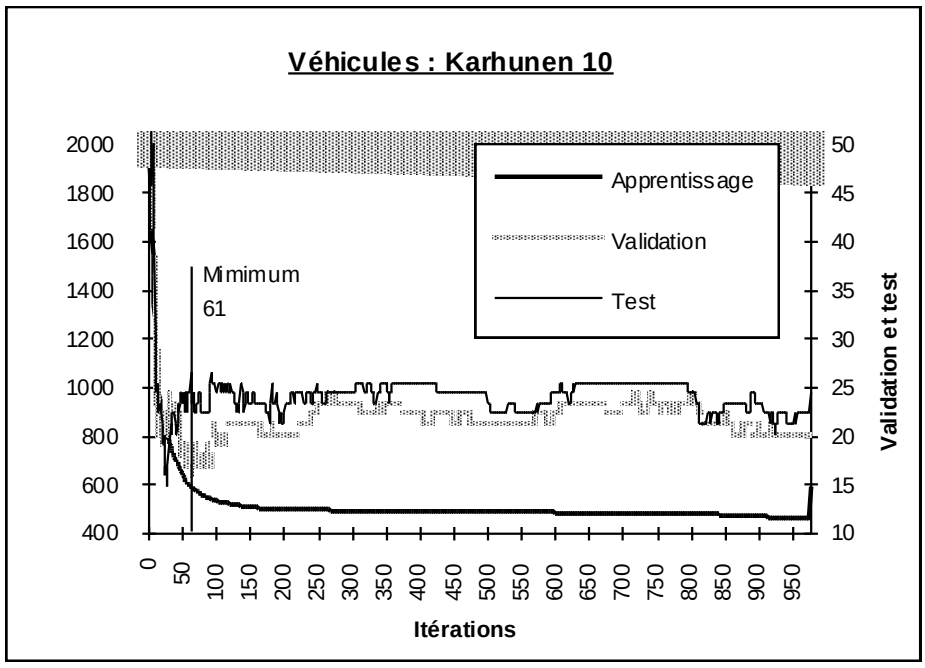


Figure 3.4

Les tables 3.11 et 3.12 décrivent les résultats obtenus lors de l'utilisation de ce réseau. Ils sont nettement moins bons que ceux obtenus sans prétraitement, mais restent honorables par rapport aux tailles de compression inférieures.

Véhicules - Karhunen 10 - Ensemble d'apprentissage							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	130	31	2	5	0	168	77.3
2	43	118	4	3	0	168	70.2
3	0	3	164	1	0	168	97.6
4	1	2	1	150	0	154	97.4

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	85.7%	83.5%	68.0%
Indécidable :	0%	4.7%	27.4%
Incorrecte :	14.3%	11.8%	4.6%

Table 3.11

Véhicules - Karhunen 10 -Ensemble de test							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	16	5	1	2	0	24	66.6
2	7	14	1	2	0	24	58.3
3	2	0	22	0	0	24	91.6
4	0	1	2	21	0	24	87.5

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	76%	73.9%	62.1%
Indécidable :	0%	3.2%	30.9%
Incorrecte :	24%	22.9%	7.0%

Table 3.12

6.3. Compressions inférieures à 10

Nous détaillons uniquement les résultats de la classification suite à une compression à 8, pour permettre une comparaison ultérieure avec la méthode NLPCA. Suite à l'application de la méthode de Karhunen, les tables 3.13 et 3.14 indiquent qu'il n'y a plus assez d'informations pour permettre au réseau de classer correctement les véhicules. La compression à 10 est une limite pour ce problème. En dessous, les résultats sont très mauvais; c'est pourquoi nous n'entrerons pas plus dans les détails. Des informations complémentaires sont reprises dans la table 3.15.

Véhicules - Karhunen 8 - Ensemble d'apprentissage							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	73	32	33	30	0	168	43.4
2	48	53	34	33	0	168	31.5
3	1	26	117	24	0	168	69.6
4	0	4	76	74	0	154	48.0

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	48.2%	33.5%	0.1%
Indécidable :	0%	32.5%	99.8%
Incorrecte :	51.8%	34.0%	0.1%

Table 3.13

Véhicules - Karhunen 8 - Ensemble de test							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	12	5	3	4	0	24	50.0
2	8	6	4	6	0	24	25.0
3	3	6	12	3	0	24	50.0
4	1	0	14	9	0	24	37.5

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	40.6%	30.2%	0%
Indécidable :	0%	26.6%	100%
Incorrecte :	59.4%	43.2%	0%

Table 3.14

Réduction à	Itération	Exactitude apprentis.	Exactitude test	Nombre itérations	Temps réel	Temps CPU
8	4	48.2%	40.6%	404	1h 07'	1h 00'
4	7	34.6%	34.1%	249	51'	25'

Table 3.15

7. Conclusions

Parmi les méthodes de décorrélation linéaire, une des transformations qui préservent l'information de manière optimale est la méthode de Karhunen-Loève. Les

variables caractéristiques, également appelées facteurs, de cette méthode sont des combinaisons linéaires des variables originales du problème. Les coefficients de cette transformation linéaire sont tels que si la transformation est appliquée à l'ensemble de données et ensuite inversée, il y aura une différence minimale (au sens des moindres carrés) entre les données originales et les données reconstruites.

Comme le montre nos essais, les données n'ont pas besoin de remplir certaines conditions pour être prétraitées. La méthode de Karhunen-Loève est valable quelle que soit la description des données. Nos simulations neurales montrent qu'elle est une solution à notre problème. Après prétraitements, les résultats de nos réseaux de neurones opérant les classifications sont toujours très fiables, mais avec un temps d'obtention nettement inférieur. Les vecteurs de données ont été réduits significativement sans perdre trop d'informations et les réseaux les utilisant ont pu diminuer leur complexité.

Bien entendu, les temps de simulation dépendent de l'importance de la compression. Si elle est faible, le réseau disposera de suffisamment de caractéristiques pour reconnaître correctement les patterns. Cependant, le nombre de neurones nécessaires impliquera un apprentissage long. Si la compression est importante, le réseau disposera de moins d'informations pour classer les vecteurs et aura un taux d'erreurs supérieur. De plus, bien que sa taille soit plus petite, le temps d'apprentissage ne le sera pas forcément. En effet, les itérations prendront chacune moins de temps, mais il en faudra peut-être beaucoup plus pour que le réseau arrive à se créer, c'est-à-dire à essayer de corriger ses erreurs. Il faut donc trouver le juste milieu. Malheureusement, il n'y a pas de règle pour le découvrir. L'expérience personnelle est importante.

CHAPITRE 4
LA MÉTHODE LPC

1. Introduction

La méthode LPC (Linear Predictive Coding) est utilisée pour compresser un signal échantillonné de sorte qu'il puisse être stocké sous une forme plus compacte. La forme originale devrait être exactement récupérable à partir de la version compressée. La méthode se base sur le fait que s'il y a redondance dans le signal, il est prédicable, avec une erreur minimale, à partir de ses valeurs précédentes et d'un petit nombre de coefficients LP. C'est l'origine du nom de cette méthode.

En réalité, nous n'utiliserons pas la méthode LPC dans sa définition première. Elle ne nous apporterait rien car la compression ne porte pas sur la réduction de la taille des patterns, dans ce cas les échantillons du signal, mais dans la réduction de la taille des valeurs contenues dans ces patterns. Cependant, nous allons montrer que les coefficients LP qu'elle utilise et choisis en nombre quelconque, sont idéaux pour représenter les signaux.

2. La méthode

Les coefficients LPC (Linear Predictive Code) sont en réalité avant tout une représentation de la puissance spectrale d'un signal.

Il existe différents estimateurs de la puissance totale d'un signal. Dans le cas discret qui nous intéresse et pour une fonction de temps, elle peut être représentée par :

$$\sum_{j=0}^{N-1}|c_j|^2 \quad (sum\ squared\ amplitude)$$

$$\frac{1}{N}\sum_{j=0}^{N-1}|c_j|^2 \quad (mean\ squared\ amplitude)$$

$$\Delta\sum_{j=0}^{N-1}|c_j|^2 \quad (time-integral\ squared\ amplitude)$$

où notre fonction c(t) a été échantillonnée tous les intervalles de temps Δ pour obtenir les N valeurs $c_0 \ldots c_{N-1}$.

Dans le domaine fréquentiel, cette estimation peut être obtenue par la somme des puissances suivantes :

$$\begin{aligned} P(f_0) &= \frac{1}{N^2}|C_0|^2 \\ P(f_k) &= \frac{1}{N^2}\left[|C_k|^2 + |C_{N-k}|^2\right] \quad k = 1,2,\ldots,\left(\tfrac{N}{2}-1\right) \\ P(f_{N/2}) &= \frac{1}{N^2}|C_{N/2}|^2 \end{aligned} \qquad (1)$$

où les f_k sont définis pour les fréquences :

$$f_k \equiv \frac{k}{N\Delta} \quad k = 0,1,\ldots,N/2$$

Commentaire [2]: ce qui correspond à l'intervalle significatif de Nyquist

et les coefficients C_k sont obtenus par la transformée de Fourier :

$$C_k = \sum_{j=0}^{N-1} c_j\, e^{2\pi ijk/N} \quad k = 0,\ldots,N-1$$

Pour se rendre compte de cette équivalence, il suffit d'appliquer la forme discrète du théorème de Parseval qui dit que :

$$\sum_{k=0}^{N-1}|h_k|^2 = \frac{1}{N}\sum_{n=0}^{N-1}|H_n|^2$$

où H est la transformée de Fourier de h.

Nous avons obtenu une fonction d'estimation de la puissance spectrale d'un processus par transformée de Fourier. Ce n'est pas la seule méthode, ni forcément la meilleure. En partant de celle-ci, nous en présentons maintenant une autre.

Si nous travaillons dans le plan z, en partant de (1) et en simplifiant les notations, par la relation $z \equiv e^{2\pi i f \Delta}$ l'estimateur FFT peut être écrit :

$$P(f) = \left| \sum_{k=-N/2}^{N/2-1} c_k z^k \right|^2 \qquad (2)$$

Ce modèle possède plusieurs noms : *"direct method"*, *"moving average (MA) model"* et *"all-zero model"* . Cette dernière dénomination vient du fait que le modèle peut avoir des zéros, mais pas de pôles. Cela nous amène à proposer une autre définition qui aurait les caractéristiques opposées :

$$P(f) \approx \frac{1}{\left| \sum_{k=-M/2}^{M/2} b_k z^k \right|^2} = \frac{a_0}{\left| 1 + \sum_{k=1}^{M} a_k z^k \right|^2} \qquad (3)$$

Les différences entre les approximations (2) et (3) ne sont pas juste cosmétiques. Ce sont des approximations possédant des caractères très différents. La propriété la plus importante est que l'estimateur (3) peut avoir des pôles, correspondant à une puissance spectrale infinie ou a un pic. A l'inverse, l'estimateur (2), qui ne peut avoir que des zéros, ne pourra que donner une approximation des pics par un polynôme. L'approximation (3) est appelée : "*all-poles model"*, *"maximum entropy method (MEM)"* ou *"autoregressive model (AR)"*.

Il reste cependant à déterminer les coefficients a_i à partir d'un ensemble de données, pour pouvoir calculer l'estimation spectrale.

Considérons l'autocorrélation à l'étape j de la fonction échantillonnée c_k, soit :

$$\phi_j \equiv \underset{i}{E}\left[c_i c_{i+j}\right] \quad j = \ldots -3, -2, -1, 0, 1, 2, 3 \ldots \tag{4}$$

où $\underset{i}{E}[\ ]$ représente la fonction moyenne sur i. Pour un nombre fini d'échantillons c_0 à c_N, l'estimation la plus naturelle de (4) est :

$$\phi_j = \phi_{-j} \approx \frac{1}{N+1-j} \sum_{i=0}^{N-j} c_i c_{i+j} \quad j = 0, \ldots, N \tag{5}$$

En d'autres termes, à partir de N+1 points de données, on peut estimer l'autocorrélation à N+1 différents niveaux.

Le théorème de Wiener-Khinchin dit que :

$$Corr(g,g) \Leftrightarrow |G(f)|^2$$

On peut en déduire que la transformée de Fourier de l'autocorrélation est égale à la puissance spectrale. L'équation qui doit dès lors être satisfaite par les coefficients de l'équation (5) est :

$$\frac{a_0}{\left|1 + \sum_{k=1}^{M} a_k z^k\right|^2} \approx \sum_{j=-M}^{M} \phi_j z^j \tag{6}$$

Il faut noter que M, le nombre de coefficients dans l'approximation à gauche du signe, peut être n'importe quel entier, supérieur, inférieur ou égal à N, le nombre total d'autocorrélations disponibles. M est appelé : "ordre" ou "nombre de pôles d'approximation".

Quelle que soit la valeur M choisie, la série du membre de gauche définit une sorte d'extrapolation de la fonction d'autocorrélation aux valeurs supérieures à M et même supérieures à N; c'est-à-dire plus grand que l'ensemble de données peut actuellement mesurer. Il peut être montré que cette méthode d'extrapolation particulière a parmi toutes les autres méthodes d'extrapolation l'entropie maximale; d'où l'appellation MEM (Maximum Entropy Method).

Étant donné que les coefficients a_k représentent très bien l'information et qu'ils peuvent être choisis en nombre quelconque, ils sont idéaux pour compresser les données de signaux.

Revenons au calcul des termes a_k. Il faut pour cela résoudre le système (6). Les termes d'autocorrélations sont calculables à partir de la fonction à représenter. Bien que cela ne soit pas évident à première vue, l'équation (6) implique un ensemble linéaire de relations entre les termes d'autocorrélations et les coefficients a_0 et a_k. En fait, les coefficients doivent satisfaire l'équation matricielle suivante :

$$\begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & \cdots & \phi_M \\ \phi_1 & \phi_0 & \phi_1 & \cdots & \phi_{M-1} \\ \phi_2 & \phi_1 & \phi_0 & \cdots & \phi_{M-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_M & \phi_{M-1} & \phi_{M-2} & \cdots & \phi_0 \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \cdots \\ a_M \end{bmatrix} = \begin{bmatrix} a_0 \\ 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix} \tag{7}$$

La première matrice de (7) est une matrice de Toeplitz symétrique; c'est-à-dire une dont les éléments sont constants le long de la diagonale. En choisissant un algorithme de résolution efficace de (7), on obtiendra alors les données demandées; dans notre cas : une compression préservant un maximum d'informations.

Dans la fin de cette section, nous allons expliquer brièvement pourquoi ces coefficients portent le nom de coefficients LPC.

Posons :

$$y_n = \sum_{j=1}^{N} d_j y_{n-j} + x_n \tag{8}$$

L'équation (8) est une filtre récursif linéaire prédisant la valeur y_n suivante à partir des N précédentes valeurs y_{n-j}, j = 1, …, N. x_n est la divergence de la prédiction pour l'étape n; c'est-à-dire la quantité qui doit être ajoutée à la valeur prédite pour obtenir la valeur correcte de y_n. Si les valeurs prédites sont d'elles-mêmes assez bonnes, alors la correction à apporter sera, en moyenne, faible; c'est-à-dire :

$$\sum_n |x_n|^2 << \sum_n |y_n|^2$$

L'idéal étant d'avoir $|x_n| << |y_n|$ pour tous les n.

Pour avoir utilité du filtre (8), il est nécessaire de trouver de bons coefficients prédictifs linéaires (LP) $d_1, \ldots, d_n$. Il apparaît alors qu'il y a une forte relation entre la prédiction linéaire et la méthode du maximum d'entropie (MEM). Les coefficients a_i calculés par MEM sont les coefficients LP; d'où leur nom.

3. Le programme

Le programme est très simple. Il n'y a pas d'options envisageables. Une fois les noms du fichier à compresser et du fichier résultat connus, ainsi que la taille de la compression, les termes d'autocorrélations sont calculés et l'équation matricielle (6) est résolue par l'algorithme de Burg qui tire profit du caractère symétrique de la matrice.

4. Analyse des phases du sommeil

4.1. Introduction

Avec la méthode de Karhunen, nous avons obtenu un taux de classification valable jusqu'à une compression de la taille des entrées à 20. A priori, puisque la méthode LPC est définie pour des signaux, une diminution à 20 semblait envisageable pour elle aussi. Nous ne nous sommes pas limité à cette taille et avons essayé des valeurs inférieures. Les deux prochaines sections décrivent les résultats.

4.2. Compression à 20

Les données prétraitées et réduites de la taille 100 à la taille 20 par la méthode LPC, ont été présentées à un perceptron. Celui-ci utilise toujours les mêmes paramètres que ceux choisis pour nos autres études des phases du sommeil. Il comprend 20 neurones en entrée, 9 en couche cachée et 3 en sortie. Il est identique au réseau utilisé après prétraitement par Karhunen. Le nombre d'exemples pour l'apprentissage est largement suffisant par rapport à sa taille.

La simulation est représentée à la figure 4.1. L'apprentissage est très rapide. 141 itérations sont nécessaires pour obtenir la convergence. La fonction d'apprentissage passe de 4 771 à 40. La fonction de validation chute de 526 à 7. A ce moment la fonction de test indique qu'il n'y a plus classification erronée qu'à moins de 2%.

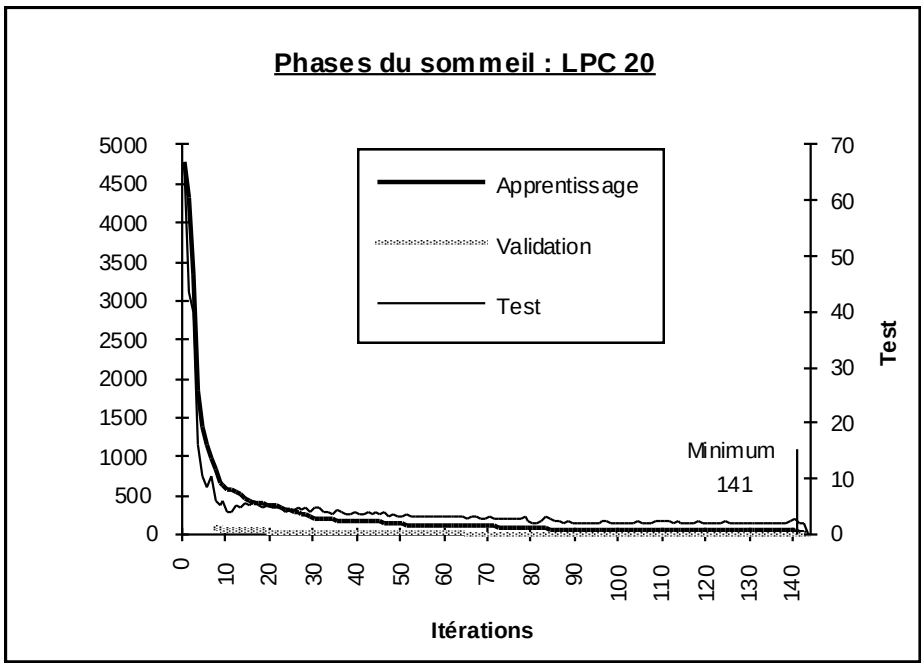


Figure 4.1

Un SUN a travaillé pendant 23 heures et 13 minutes pour obtenir ces 141 itérations. En temps CPU, cela n'a pris que 5 heures et 49 minutes. De plus, notons la facilité du réseau à s'adapter; les principales variations se sont produites dans les 20 premières itérations.

Le meilleur réseau a été sauvé à la fin de la simulation. Son utilisation nous a fourni les résultats des tables 4.1 et 4.2. L'apprentissage n'aurait pu mieux se passer. Pour les exemples, un résultat de 99.7% de classification correcte est exceptionnel. La classe la moins bien reconnue, la classe deux, l'est quand même à 99.5%. La classe un atteint le maximum de 100% de reconnaissance!

Phases du sommeil - LPC 20 - Ensemble d'apprentissage						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	1800	0	0	0	1800	100.0
2	0	1791	9	0	1800	99.5
3	0	7	1793	0	1800	99.6
Classification	Libre		Distance 0.2		Différence 0.5	
Correcte :	99.7%		99.6%		99.7%	
Indécidable :	0%		0.2%		0.1%	
Incorrecte :	0.3%		0.2%		0.2%	

Table 4.1

Pour l'ensemble de test, les résultats restent exceptionnels : 98.1% de classification correcte. Ils sont bien entendus légèrement inférieurs à ceux des données d'apprentissage, mais de peu. On peut remarquer que les classes un et trois sont reconnues parfaitement. Seule la classe deux donne des résultats inférieurs avec 94.2%. Une solution pour tenter de remédier à ce problème serait d'augmenter le nombre d'exemples de la classe deux dans les données d'apprentissage. Mais même ainsi, notre réseau est fiable.

Phases du sommeil - LPC 20 -Ensemble de test						
Classe désirée	Classe obtenue				∑ /classe	% correct
	1	2	3	?		
1	1000	0	0	0	1000	100.0
2	1	942	57	0	1000	94.2
3	0	0	1000	0	1000	100.0
Classification	Libre		Distance 0.2		Différence 0.5	
Correcte :	98.1%		98.0%		97.9%	
Indécidable :	0%		0.8%		0.2%	
Incorrecte :	1.9%		1.2%		1.9%	

Table 4.2

Avec un taux de réussite aussi élevé, on n'est pas étonné que les critères plus restrictifs de classification (distance et différence) sont inutilisables. Les résultats sont fiables.

4.3. Compression à 10

Étant donné les excellents résultats d'une compression à 20 par la méthode LPC, nous avons essayé d'aller plus loin. Nous avons réduit nos vecteurs initiaux par la même méthode jusqu'à la taille 10. Nous les avons ensuite présentés au même réseau que celui utilisé après compression à 10 par Karhunen; c'est-à-dire une architecture de 10 entrées, 5 neurones en couche cachée et 3 sorties. Les autres définitions du réseau restant semblables à celles utilisées jusqu'à maintenant.

La figure 4.2 décrit l'apprentissage de ce réseau. Le résultat est surprenant. Le réseau est toujours parfaitement capable d'apprendre la différence entre les phases du sommeil. Il y a convergence en 992 itérations. La fonction d'apprentissage est passée de 3 975 à 39. La fonction de validation, partant de 441, atteint la valeur 8. Le pourcentage d'erreur sur l'ensemble test est de 2.6% à la fin de la simulation.

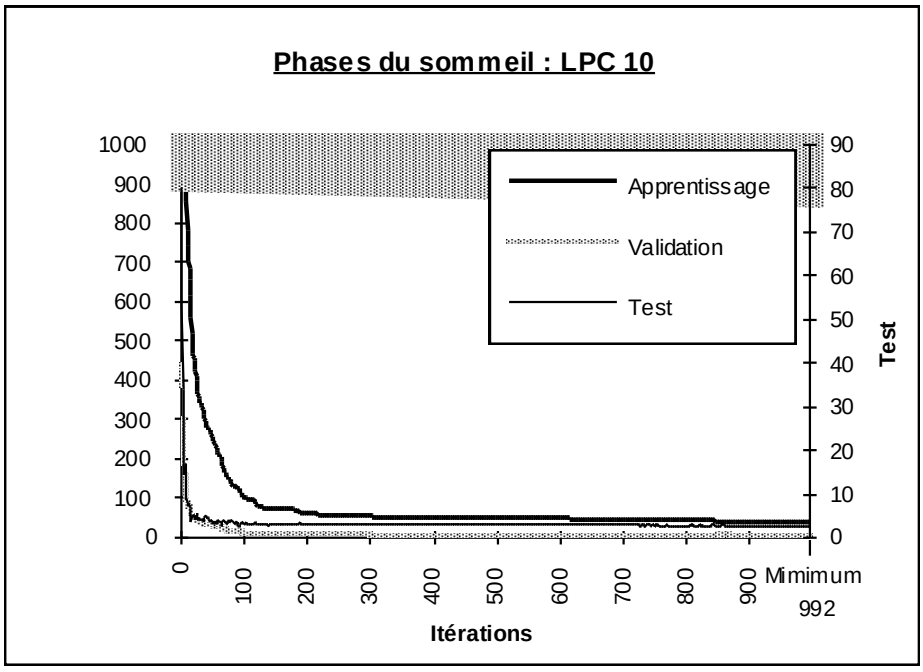


Figure 4.2

Cependant, cette taille de compression n'est pas intéressante selon nos critères de temps. En effet, le réseau a beaucoup plus de mal à apprendre la distinction

des classes. 992 itérations ont été nécessaires. Notre ordinateur a du calculé pendant 52 heures et 44 minutes. En temps CPU, les 17 heures et 30 minutes restent énormes par rapport aux résultats de la compression à 20. Remarquons cependant qu'ici aussi, une vingtaine d'itérations suffisent déjà à donner un réseau classant très bien l'ensemble de test.

Pour le réseau optimum, les résultats détaillées sont repris dans les tables 4.3 et 4.4. Les pourcentages de classification correcte pour chaque classe restent pratiquement identiques à ceux obtenus pour une compression à 20. Ils sont toujours aussi fiable. Seule la classe deux se dégrade encore un peu, tout en restant acceptable. La structure du réseau ayant changé, on peut supposer que le problème vient bien d'un manque d'exemples pour cette classe. Augmenter leur nombre ne peut qu'améliorer les choses.

Phases du sommeil - LPC 10 - Ensemble d'apprentissage						
Classe	Classe obtenue				∑	%
désirée	1	2	3	?	/classe	correct
1	1800	0	0	0	1800	100.0
2	0	1788	12	0	1800	99.3
3	0	8	1792	0	1800	99.6

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	99.6%	99.6%	99.6%
Indécidable :	0%	0.1%	0.1%
Incorrecte :	0.4%	0.3%	0.3%

Table 4.3

D'autres essais ont été réalisés avec des tailles de compression inférieures. Cependant, même avec beaucoup de temps, le réseau n'est plus capable de s'adapter. La réduction à la dimension 10 est une limite.

Phases du sommeil - LPC 10 - Ensemble de test						
Classe	Classe obtenue				∑	%
désirée	1	2	3	?	/classe	correct
1	1000	0	0	0	1000	100.0
2	0	923	77	0	1000	92.3
3	0	1	999	0	1000	99.9

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	97.4%	97.3%	97.1%
Indécidable :	0%	0.3%	0.3%
Incorrecte :	2.6%	2.4%	2.6%

Table 4.4

5. Reconnaissance des caractères manuscrits

5.1. Introduction

Nous nous sommes basé sur les résultats obtenus par la méthode de Karhunen et par les algorithmes du projet StatLog. Notre première simulation a donc débuté avec des vecteurs de données réduits à la taille 40. Peu d'autres essais ont été réalisés étant donné les premiers résultats obtenus.

5.2. Compression à 40

Une fois les données ramenées à 40 composantes, elles ont été présentées à un réseau traditionnel de 3 couches de neurones : 40 en entrée, 16 en couche cachée et 10 en sortie. Les paramètres sont à nouveau ceux de la table 2.1. Ce réseau est identique à celui utilisé après compression à 40 par Karhunen.

On voit dans la figure 4.3 la stabilité, mais aussi la limite de l'apprentissage. La fonction de validation atteint son minimum à l'itération 1 184 avec la valeur de 22% d'erreurs. La fonction de test indique alors 25%. La fonction d'apprentissage a eu le temps de chuter de 29 687 à 3 360. Le réseau s'entraîne encore pendant 89 itérations avant d'atteindre sa limite. La fonction d'apprentissage ne peut descendre en dessous de 3 358.

Ces résultats ont été obtenus après 11 jours de simulation, soit un temps réel de 263 heures et 12 minutes et un temps CPU de 170 heures et 31 minutes. Les courbes sont très régulières. Après 300 itérations, elles sont toutes les trois déjà pratiquement parallèles à l'axe des abscisses.

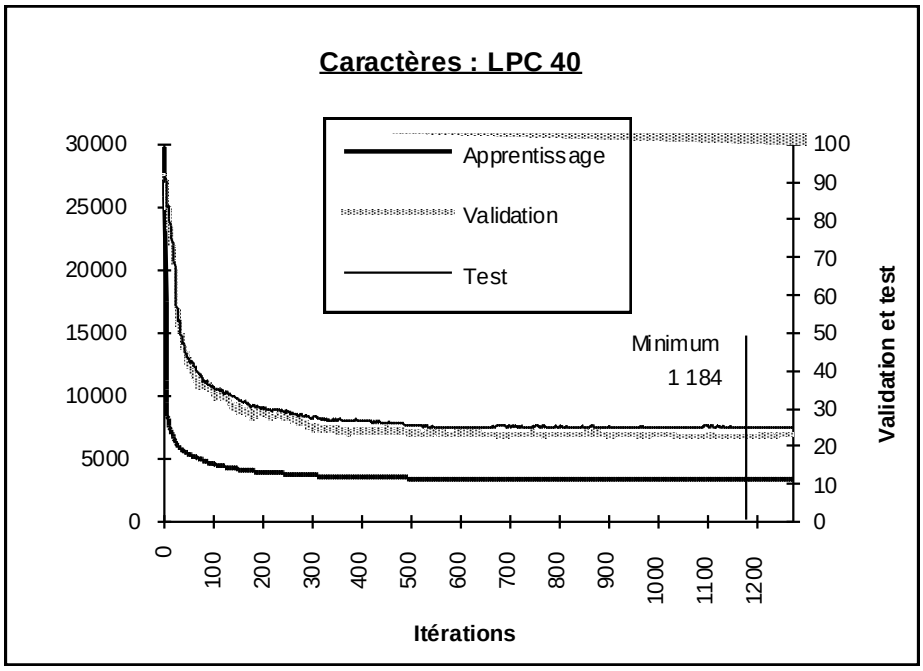


Figure 4.3

La classification détaillée obtenue à partir du meilleur réseau et reproduite dans les tables 4.5 et 4.6, indique que cette compression est uniformément mauvaise pour toutes les classes. La méthode LPC ne donne pas d'aussi bons résultats que pour nos ensembles de signaux. Avec 25% d'erreurs sur un ensemble test, ce réseau n'est pas fiable. Cela n'est pas du à un problème d'incapacité de généralisation, puisque le taux d'erreur sur l'ensemble d'apprentissage est sensiblement le même. Le taux de compression est trop élevé.

Remarquons qu'ici les critères de sélection supplémentaires sont plus actifs. Le réseau n'a pas réussi à tirer des conclusions très précises de l'ensemble d'apprentissage qu'il a du étudier.

Caractères - LPC 40 - Ensemble d'apprentissage													
Classe	Classe obtenue											∑	%
	1	2	3	4	5	6	7	8	9	10	?		
1	702	23	2	0	25	0	22	2	12	22	0	810	86.7
2	17	755	2	1	27	0	3	4	1	0	0	810	93.2
3	66	1	402	47	14	31	111	38	38	62	0	810	49.6
4	0	0	15	654	0	59	12	6	22	42	0	810	80.7
5	26	3	6	0	694	10	12	42	4	13	0	810	85.7
6	2	0	2	45	10	685	6	20	10	30	0	810	84.6
7	37	2	38	11	14	11	517	9	50	121	0	810	63.8
8	5	0	24	5	65	57	15	634	2	3	0	810	78.3
9	18	3	3	36	3	2	30	0	681	34	0	810	84.1
10	33	0	14	28	34	23	116	11	41	510	0	810	63.0

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	77%	66.7%	58.6%
Indécidable :	0%	21.8%	34.5%
Incorrecte :	23%	11.5%	6.9%

Table 4.5

Caractères - LPC 40 - Ensemble de test													
Classe	Classe obtenue											∑	%
	1	2	3	4	5	6	7	8	9	10	?		
1	260	9	1	0	9	0	5	1	6	9	0	300	86.7
2	13	278	0	0	7	0	1	0	1	0	0	300	92.7
3	22	0	133	21	6	19	49	16	11	23	0	300	44.3
4	0	0	9	234	0	26	1	5	6	19	0	300	78.0
5	8	2	4	0	251	2	5	21	3	4	0	300	83.7
6	0	0	3	26	7	232	2	14	2	14	0	300	77.3
7	13	1	16	4	8	6	187	4	13	48	0	300	62.3
8	4	0	6	4	25	19	4	236	0	2	0	300	78.7
9	11	2	2	8	1	0	7	0	245	24	0	300	81.7
10	8	0	1	15	9	7	41	8	19	192	0	300	64.0

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	74.9%	64.9%	57.3%
Indécidable :	0%	22.4%	34.8%
Incorrecte :	25.1%	12.7%	7.9%

Table 4.6

5.3. Autres tentatives

Nous avons étudié deux autres types de compression : la compression à 40 suivie d'un recentrage et la compression à 30. Les simulations utilisant ces données n'ont pas eu un comportement encourageant; c'est pourquoi nous ne les avons pas laissé se terminer. Nous sommes convaincu que le temps n'aurait pas permis d'améliorer cet apprentissage. Les courbes étaient devenus pratiquement planes. La table 4.7, indique les résultats obtenus lors de l'interruption.

Méthode	Itération	Exactitude apprentis.	Exactitude test	Nombre itérations	Temps réel	Temps CPU
40 recentré	150		67.0%	151	47h 36'	46h 52'
30	779		64.0%	782	247h 38'	223h 12'

Table 4.7

6. Reconnaissance des véhicules

Trois compressions par la méthode LPC ont été essayées pour le problème de la reconnaissance des véhicules. Rappelons qu'avec 18 entrées, le réseau utilisant les données originales fournit rapidement les résultats. Une compression n'est pas nécessaire. Le but de nos tentatives est de mesurer les capacités de la transformation LPC et d'établir des comparaisons avec les autres méthodes.

Comme le montre la table 4.8, aucun de nos essais n'a été concluant. LPC n'est pas efficace pour ce type de données. Une autre méthode est nécessaire si on désire réduire la taille du réseau. Nous n'étudierons donc pas plus en détail ces résultats.

Méthode	Itération	Exactitude apprentis.	Exactitude test	Nombre itérations	Temps réel	Temps CPU
10	87	54.6%	44.7%	331	1h 49'	1h 48'
8	62	57.3%	54.1%	610	3h 03'	1h 31'
4	120	48.3%	47.2%	254	28'	26'

Table 4.8

7. Conclusions

Les coefficients LP sont idéaux pour extrapoler des signaux. Ils possèdent cette propriété car ils correspondent à une représentation de la puissance spectrale du signal. Quelques coefficients seulement suffisent à capturer les informations. De nouveau, il n'est pas possible à priori de déterminer le meilleur nombre. L'expérience montre que des vecteurs de taille 1 000 ou 10 000 peuvent être réduits à la taille 10, 20 ou 50 selon les besoins.

Dans nos problèmes de classification, nous disposions d'un ensemble de données de type signal : les analyses du sommeil. On constate que les résultats que nous avons obtenus pour ce problème sont optimaux. La reconnaissance des caractères et des véhicules n'a pas été aussi bonne.

Comme pour le chapitre précédent, nous constatons qu'il faut trouver un juste milieu pour l'importance de la compression. Si elle trop faible, les itérations du réseau seront longues. Si elle est trop forte, le réseau reconnaîtra moins bien et réclamera peut-être plus d'itérations pour son apprentissage. Dans le cadre de la reconnaissance des phases du sommeil, des réductions à 20 et à 10 étaient acceptables pour la méthode LPC. Cependant, notre problème de classification préfère la compression la moins importante à 20, car elle permet de réduire la complexité du réseau ainsi que le nombre d'itérations nécessaires à son apprentissage.

CHAPITRE 5
LA MÉTHODE NLPCA

1. Introduction

Dans le chapitre trois, nous avons présenté une méthode de compression linéaire : celle de Karhunen-Loève. Elle consistait en un mapping linéaire d'un vecteur de données Y en un vecteur réduit T par une matrice de transformation P :

$$T = Y\,P$$

La différence principale entre la méthode de Karhunen et NLPCA (Non Linear Principal Composant Analysis) est que cette dernière permet des transformations non linéaires entre l'espace original et l'espace réduit. Si des corrélations non linéaires existent entre les variables, NLPCA décrira les données avec une plus grande précision et/ou avec moins de facteurs que Karhunen, pour autant qu'il y ait des données en suffisance pour utiliser ce mapping plus complexe.

Cette dernière remarque est importante. Notre objectif dans ce travail est de réduire la taille des données pour pouvoir utiliser un réseau plus petit et plus facile à entraîner. Comme nous le verrons, la méthode NLPCA a un effet contraire. Sa mise en oeuvre nécessite un réseau de neurones plus complexe que celui utilisé sans compression préalable, pour traiter le problème posé. Cette méthode n'est applicable que pour de petits problèmes. Étant donné sa réputation, nous la décrivons quand même.

2. La méthode

2.1. Le principe

Dans la méthode NLPCA, les transformations vers l'espace des caractéristiques est généralisé pour autoriser les relations non linéaires. Cela peut être représenté par la transformation suivante :

$$T = G(Y) \tag{1}$$

où Y est le vecteur de données dont on cherche les caractéristiques, T est le vecteur résultat compressé et G = {G_1, G_2, …, G_f} est un vecteur de f fonctions non linéaires tel que si T_i est le $i^{ème}$ élément de T,

$$T_i = G_i(Y) \tag{2}$$

La transformation inverse, restaurant la dimension originale des données, est implémentée par un second vecteur de fonctions non linéaires H = {H_1, H_2, …, H_m} :

$$Y_j^{'} = H_j(T) \tag{3}$$

La perte d'information est mesurée par E = Y - Y', et on doit donc chercher des fonctions G et H qui minimisent ||E||. Cela correspond au critère optimal de la méthode de Karhunen.

2.2. Recherche des vecteurs de fonctions G et H

Pour générer G et H, une approche fonctionnelle de base est utilisée. Cybenko, en 1989, a montré que des fonctions de la forme

$$v_k = \sum_{j=1}^{N_2} w_{jk\,2}\,\sigma\Big(\sum_{i=1}^{N_1} w_{ij1} u_i + \theta_{j1}\Big) \tag{4}$$

où σ(x) est une fonction quelconque continue et monotone croissante telle que $\sigma(x)\to 1$ lorsque $x\to\infty$ et $\sigma(x)\to 0$ lorsque $x\to-\infty$, sont capables de s'ajuster pour représenter n'importe quelle fonction non linéaire v = f(u) et avec un degré de précision arbitraire. Une fonction σ couramment choisie est la sigmoïde, définie par :

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{5}$$

Comme nous l'avons vu au chapitre 1, l'expression (4) correspond à un perceptron à trois couches :

- La couche 1, comportant N_1 neurones et dont le seul rôle est l'interfaçage entre le réseau et l'environnement, est la couche fictive d'entrée.
- La couche 2, la couche cachée, comporte N_2 neurones qui calculent

$$o_{j2} = \sigma(\sum_{i=1}^{N_1} w_{ij1} o_{i1} + \theta_{j1})$$

où o_{lm} est la sortie du neurone l de la couche m, avec dans ce cas $o_{i1} = u_i$ et w_{ijk} est le poids de la connexion allant du neurone i de la couche k au neurone j de la couche k+1.

- La couche 3, comportant pour chaque k un neurone calculant la somme de ses entrées par

$$o_{k3} = \sum_{j=1}^{N_2} w_{jk2} o_{j2}$$

avec dans ce cas $v_k = o_{k3}$, est la couche de sortie.

En pratique, deux modifications sont souvent apportées. Premièrement, plutôt que d'utiliser une fonction linéaire en sortie, on utilise des fonctions limitant les sorties à un certain intervalle fixé et fini. Deuxièmement, la fonction sigmoïde peut changer d'échelle ou être translatée sans perte de généralité pour le réseau. Ceci est utile car les ensembles de données à traiter que l'on rencontre sont souvent centrés. Dans ce travail, nous avons dès lors utilisé, pour tous nos apprentissages, la fonction tangente hyperbolique qui répond à ces conditions et qui a fait ses preuves.

On peut maintenant facilement définir les fonctions G et H. Ce sont des réseaux tels qu'on vient de les définir. Définissons G. Si m est la taille du vecteur original Y et f la taille du vecteur T des caractéristiques, N_1 vaut m et il y a f neurones en sortie (k varie de 1 à f). Étant donné que la couche cachée est là pour capturer les relations non linéaires, pour obtenir f caractéristiques indépendantes, N_2 ne doit pas être inférieur au nombre f. La fonction G_k, représentant le $k^{ème}$ facteur non linéaire, est alors définie par la formule (4). Le réseau G est représenté à la figure 5.1.a.

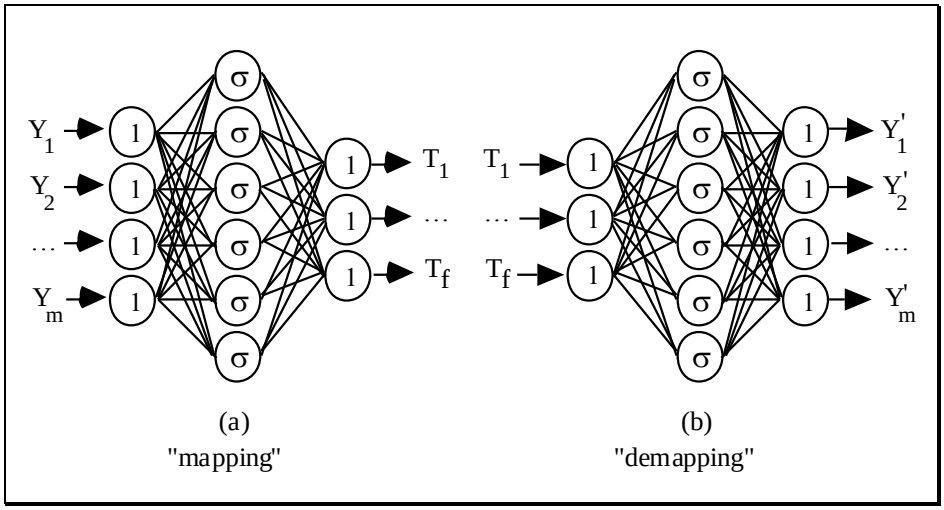


Figure 5.1

H s'obtient similairement. La couche d'entrée reçoit les f caractéristiques (N_1 = f) et dans la couche de sortie, pour retrouver les m Y_i originaux, il y a m neurones de sortie Y'_i. La couche de "demapping" contient les N_2 neurones (N_2 > f) à fonctions sigmoïdes. Le réseau type de demapping est présenté à la figure 5.1.b.

La capacité de ces réseaux à s'ajuster à une fonction non linéaire provient de la présence de fonctions d'activations non linéaires en couche cachée. En effet, sans la présence de neurones non linéaires en couche cachée, ces réseaux seraient seulement capables de produire des combinaisons linéaires des entrées à la sortie, ce qui n'est pas suffisant pour notre problème.

Commentaire [3]: cette position de sigma permet la compression non linéaire

Une autre position ou l'absence de sigma implique une relation linéaire

Pour pouvoir utiliser les fonctions G et H, deux conditions sont à remplir. Il faut décider du nombre de neurones dans les couches cachées. Pour éviter les ambiguïtés, rebaptisons le N_2 du réseau G en M_1 (M pour Mapping) et le N_2 du réseau H en M_2. Malheureusement, il n'y a pas de loi précise permettant de fixer le nombre de neurones dans les couches cachées d'un réseau. Dans notre cas de compression, nous ne pourrons que fixer des bornes et une estimation. Nous devons également découvrir les meilleurs coefficients W_{ijk} de la formule (4). Cela revient à réaliser l'apprentissage de nos réseaux.

2.3. L'apprentissage des réseaux

Il y a un problème. En effet, pour réaliser l'apprentissage, on doit fournir au réseau des vecteurs d'entrées **et** les vecteurs de sorties correspondants. Malheureusement, dans le cas du réseau G, on connaît les entrées (Y_i), mais pas les sorties et dans le cas du réseau H, on connaît les sorties souhaitées (Y_i), mais pas les entrées. Pour résoudre ce problème, il suffit de se rendre compte que les entrées de H sont les sorties de G. En combinant les deux réseaux, on en obtient un dont les entrées **et** les sorties sont connues. On peut donc réaliser son apprentissage. Ce réseau correspond à la fonction identité. Il est représenté à la figure 5.2.

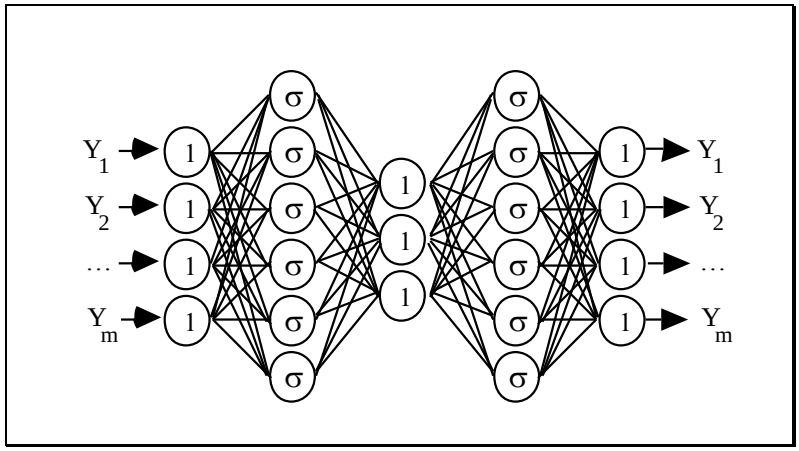


Figure 5.2

Le réseau de la figure 5.2 comprend trois couches cachées : la couche de mapping de G, la couche centrale dont les sorties correspondent aux caractéristiques T et la couche de demapping de H. La seconde couche cachée est appelée couche de compression à cause de sa dimensionnalité inférieure.

Lors de l'apprentissage, les poids sont modifiés de manière à minimiser la différences entre les sorties obtenues Y'i et les sorties attendues Yi pour tous les vecteurs présentés. L'apprentissage est terminé lorsque E, la somme des carrés des erreurs, est minimum; c'est-à-dire pour n vecteurs de données :

$$E = \min_{w_{jkl}} \sum_{p=1}^{n} \sum_{i=1}^{m} \left(Y_i - Y_i^{'} \right)_p^2 \qquad (6)$$

E est le carré de ||matrice d'erreurs||, le critère d'optimalité de la méthode de Karhunen. Dès lors, minimiser E durant l'apprentissage résulte en une minimisation de la perte d'information au même sens que Karhunen.

Après l'apprentissage, le réseau combiné n'est plus nécessaire et peut être désagrégé en deux réseaux : G et H. G est la fonction d'intérêt. Les données sont propagées à travers G pour projeter les données dans l'espace de dimensionnalité inférieur des caractéristiques.

2.4. Détermination de la taille des couches

Dans le réseau combiné, il y a m noeuds d'entrées et de sorties et f noeuds dans la couche de compression. Cependant, il n'existe pas de méthode définitive pour décider à priori de la dimension des couches de mapping et de demapping (nous les appellerons parfois les deux couches de mapping).

Le nombre de noeuds de mapping est lié à la complexité des fonctions linéaires qui peuvent être générées par le réseau. S'il y a trop peu de noeuds de mapping, l'ajustement risque d'être faible à cause de la capacité de représentation faible du réseau. Cependant, si ce nombre de noeuds est trop élevé, le réseau risque d'apprendre les variations stochastiques des données plutôt que les fonctions sous-jacentes.

L'approche la plus simple à ce problème consiste dès lors à limiter le nombre de poids dans le réseau à une fraction du nombre de contraintes imposées par les données. Pour chaque vecteur de données, une contrainte séparée est imposée par chaque noeud de sortie. Le nombre de paramètres ajustables doit donc être inférieur à n*m. Pour le réseau combiné, en assumant que tous les noeuds ont des bias (le paramètre θ), le nombre de paramètres ajustables vaut $(m+f+1)*(M_1+M_2)+m+f$. Ces deux constatations impliquent les inégalités uivantes :

Commentaire [4]: m*M1+M1*f+f*M2 +M2*m
= (m+f) (M1+M2)
= nombre de poids w

M1+f+M2+m
= nombre de bias

+ -> = (m+f+1) (M1+M2) + m + f

$$\begin{aligned} &(m+f+1)(M_1+M_2)+m+f << mn \\ &\Leftrightarrow M_1+M_2 << \frac{m(n-f)}{m+f+1} \end{aligned} \qquad (7)$$

Pour un petit nombre de facteurs (f << m et n), cette expression peut être approximée par :

$$M_1 + M_2 << n \tag{8}$$

De plus, si le nombre de noeuds de mapping ou de demapping autorisé par les inégalités (7) et (8) est inférieur à f, alors il n'y a pas assez de données pour supporter l'extraction de f facteurs non linéaires, puisque la couche de compression apparaît, par définition, dans la seconde couche cachée du réseau combiné, entre les deux couches de mapping.

Rappelons enfin la règle des 10%. Un apprentissage est facilité s'il y a au moins dix fois plus de vecteurs de données que de synapses dans le réseau; c'est-à-dire :

$$n > 10(m + f)(M_1 + M_2) \tag{9}$$

2.5. Remarques sur les couches cachées

On a vu qu'un réseau combiné de trois couches cachées permettait la construction d'un réseau de compression. Est-ce que trois couches sont nécessaires ? Imaginons que l'on supprime les couches de mapping; ne laissant que la couche de compression. Si les fonctions de cette couche sont linéaires, le réseau correspond en fait à la méthode de Karhunen. Cela a été montré par Sanger en 1989. Si les fonctions sont des sigmoïdes, les fonctions G et H sont fortement restreintes; seules des combinaisons linéaires des entrées, compressées par la sigmoïde dans son intervalle de variation, peuvent être présentées. Dès lors, les résultats ne sont pas souvent meilleurs que ceux obtenus par la méthode de Karhunen.

La structure à cinq couches dont trois cachées est donc la meilleure.

3. Le programme

Comme il vient d'être dit, la méthode NLPCA consiste en l'apprentissage d'un réseau de neurones. Nous avons donc utilisé le simulateur de réseaux de neurones de Monsieur Fombellida. Cependant, différents outils supplémentaires ont été nécessaires. Nous avons donc programmé un module supplémentaire.

Le menu général se présente comme suit :

```
          +++ OUTILS POUR LA METHODE DE COMPRESSION NLPCA +++

0. Aide

1. Conversion des donnees au format NLPCA
2. Selection des premieres couches d'un reseau
3. Modification du nombre de sorties d'un fichier de donnees
4. Utilisation type d'un reseau de SIRENE
5. Recuperation des sorties d'un reseau

9. Retour au menu principal
```

3.1. Conversion des données au format NLPCA.

La première étape dans la méthode NLPCA est de construire le réseau combiné et de réaliser son apprentissage. Lorsque sa structure est définie, il faut fournir au simulateur un fichier de données tel que, pour chaque pattern, les sorties souhaitées soient identiques aux entrées.

Le premier sous-menu réalise la conversion d'un fichier d'apprentissage au format standard de SIRENE en un fichier "identité" au même format.

3.2. Sélection des premières couches d'un réseau.

Lorsque l'apprentissage du réseau combiné est terminé, on n'a plus besoin que des premières couches, celles qui correspondent au réseau de compression G.

Le second sous-menu transforme le réseau sauvé par SIRENE lors de l'apprentissage, de manière à récupérer les X premières couches et obtenir un réseau de compression utilisable dans SIRENE.

3.3. Modification du nombre de sorties et utilisation type

Ces commandes sont déjà présentes dans le module d'outils décrit au chapitre deux. Elles réapparaissent ici, car la méthode NLPCA nécessite plusieurs

utilisations du réseau de compression et souvent avec des fichiers de données qui n'ont pas à l'origine le bon format.

3.4. Récupération des sorties d'un réseau

Dans notre cas, la méthode NLPCA n'est qu'un prétraitement. Les vecteurs comprimés seront réutilisés par un autre réseau de neurones dans un but à déterminer. Il faut donc que ces premiers résultats soient utilisables par SIRENE, c'est-à-dire qu'ils soient écrits dans un fichier au format standard.

Le quatrième sous-menu récupère les sorties du réseau de compression à partir du fichier "use.txt" généré par SIRENE, ainsi que les sorties souhaitées associées aux vecteurs non comprimés. Le résultat est un fichier standard d'apprentissage.

4. Les traitements

La démarche comporte plusieurs étapes :

1. Convertir les fichiers de données au format NLPCA.
2. Réaliser un apprentissage avec ces données.
3. Récupérer les premières couches du meilleur réseau obtenu.
4. Utiliser le nouveau réseau pour les trois ensembles de données.
5. Récupérer les résultats de "use.txt" et les convertir en fichier de données SIRENE.
6. Réaliser un apprentissage d'un nouveau réseau avec ces derniers fichiers pour résoudre le problème posé.
7. Analyser les résultats.

Détaillons ces étapes. Pour réaliser l'apprentissage du réseau combiné, il faut fournir des fichiers de données tels que les vecteurs des sorties souhaitées soient identiques aux vecteurs des entrées. La première étape est donc la conversion de nos fichiers originaux à ce format. Cela est réalisé par la première commande de notre programme.

Il suit ensuite l'apprentissage de ce réseau. L'utilisateur est libre de choisir ses paramètres, mais il peut aussi utiliser le fichier type de commandes créé dans le module "outils" de notre travail.

L'apprentissage réalisé, le réseau combiné n'est plus nécessaire. On ne doit plus conserver que la première moitié : celle qui correspond à la compression. Cette opération est réalisée par la deuxième commande du programme.

Une fois le réseau de compression disponible, on doit l'utiliser pour obtenir les vecteurs réduits de données. Un problème se pose. Le fichier nécessaire lors de cette utilisation doit contenir des vecteurs de sorties comprenant autant de valeurs qu'il y a de sorties au réseau. Cela pour respecter les normes de SIRENE. Cela n'est en général pas le cas, car les fichiers à notre disposition comprennent une valeur de sortie par classe et il n'y a pas de rapport entre la taille de la compression (le nombre de sorties du réseau) et le nombre de classes. Il est donc la plupart du temps nécessaire de modifier la taille des vecteurs de sorties des fichiers. On peut le réaliser avec la troisième ou la quatrième commande de notre menu. La quatrième à l'avantage de ne faire cette modification que si nécessaire et de fournir un fichier type d'instructions pour l'utilisation du réseau. La simulation peut avoir lieu.

On obtient un fichier "use.txt" comprenant tous les résultats de l'utilisation du réseau. Ce fichier n'est pas un fichier de données pouvant servir à l'apprentissage, la validation ou le test. Pour pouvoir utiliser les vecteurs comprimés dans un réseau de neurones, il faut récupérer ces vecteurs dans "use.txt" et les vecteurs des sorties attendues correspondantes dans les fichiers de départ. Le résultat doit être au format de SIRENE. Cette opération est réalisée par la cinquième commande de notre module. Ceci clôture la phase de compression.

Il ne reste plus qu'à utiliser nos nouveaux ensembles de données pour résoudre le problème posé et à analyser les résultats obtenus.

Comme on peut le voir, il y a beaucoup d'options envisageables. Il n'était pas possible de réaliser une commande unique s'occupant de toute la compression NLPCA. Notre programme laisse toute sa liberté à l'utilisateur, mais lui offre de nombreux raccourcis.

Nous pouvons maintenant passer à l'application de cette méthode pour les problèmes décrits au chapitre deux.

5. Les phases du sommeil et les caractères manuscrits

Le but de ce travail est de trouver des méthodes pour diminuer la taille des réseaux et accélérer leur apprentissage. L'algorithme de compression NLPCA nécessite, comme on l'a vu, un réseau de neurones d'au moins 5 couches et comportant autant de sorties et d'entrées qu'il y a de données dans un vecteur du problème. Son apprentissage est inimaginable pour des vecteurs de données de grande taille.

Pour le problème de la reconnaissance des caractères, la méthode de Karhunen a donné de bons résultats pour une taille réduite à 40. Pour obtenir ce taux de compression par NLPCA, il faudrait un réseau comprenant au grand minimum 5 320 synapses (256 * 10 + 10 * 10 + 10 * 10 + 10 * 256). Pour respecter la règle des dix pourcents, nous devrions également disposer d'au moins 53 200 exemples d'apprentissage. Malheureusement, nous n'en avons que 8 100. Même si nous supposions avoir assez d'exemples, le temps que nécessiterait l'apprentissage d'un tel réseau serait phénoménal. Notons de plus que le problème de classification sans prétraitement n'a besoin que d'un réseau plus petit (au minimum de 2 660 synapses). C'est le même problème pour les phases du sommeil. Avec des vecteurs de 100 données, la méthode NLPCA n'est pas applicable. Elle nécessite trop de ressources.

6. Reconnaissance des véhicules

6.1. Introduction

Le problème de la reconnaissance des véhicules est envisageable par la méthode NLPCA. Avec 18 données par pattern, le réseau combiné à créer reste réalisable. Notons cependant dès maintenant que le peu de patterns disponibles sera une limite à son utilisation.

Comme précédemment, les réductions à 10, 8 et 4 ont été envisagées. Les réseaux combinés ont été testés avec différents nombres de neurones en couches cachées.

6.2. Compression à 10

Nous avons réalisé chacune des étapes décrites dans la section 4. Pour le réseau combiné, nous avons choisi une architecture de cinq couches comprenant par ordre de propagation 18, 12, 10, 12 et 18 neurones. Il y a donc 672 synapses. Les paramètres utilisés sont les mêmes que d'habitude : ceux de la table 2.1.

L'apprentissage du réseau combiné à partir des 658 exemples de l'ensemble de données, est illustré par la figure 5.3. La fonction d'apprentissage débute à 3 103 et converge vers la valeur 24. Cela nécessite 85 itérations. En ce court laps de temps, la fonction de validation atteint un minimum d'erreurs de 26.6%. La fonction de test indique 13.8%. Le réseau a donc parfaitement réussi à compresser les données d'apprentissage, mais n'a pas réussi à généraliser parfaitement ses résultats. Ce n'est pas étonnant, vu le faible nombre d'exemples présentés (658) comparé au nombre de synapses (672). Nous sommes loin de la règle des dix pourcents.

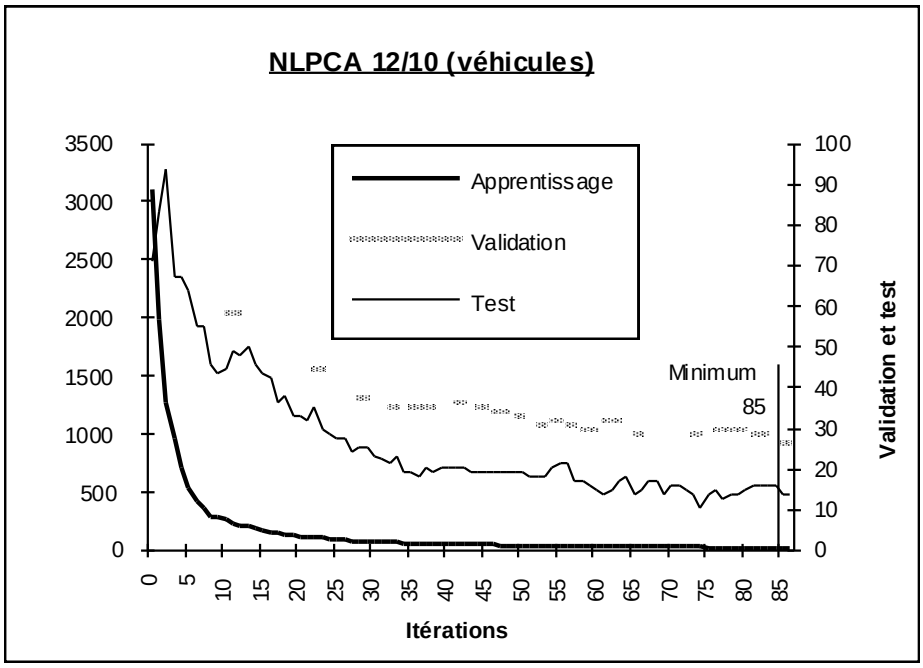


Figure 5.3

La phase de compression a nécessité 3 heures et 36 minutes. En temps CPU, cela a pris 1 heure et 42 minutes. Les courbes ont toujours la même allure que d'habitude, mais étant donné le faible nombre d'itérations avant la convergence, on a l'impression que les courbes de test et de validation n'ont pas eu le temps d'atteindre leur minimum. Il serait intéressant de recommencer cette simulation avec plus d'exemples d'apprentissage.

Une fois les vecteurs réduits récupérés, ils ont été présentés au réseau se chargeant de la reconnaissance des véhicules. Ce réseau comportait 10 neurones en entrée, 7 en couche cachée et 4 en sortie. Les paramètres restent les mêmes. Cette fois-ci, on dispose de 658 exemples pour 98 synapses. L'équilibre se rétablit presque.

La figure 5.4 représente l'étude du réseau. La fonction de validation atteint son minimum très vite : 23% d'erreurs. 24 itérations ont été nécessaires. Après, elle et la fonction de test ne varient presque plus. La fonction de test indique 22% d'erreurs. La fonction d'apprentissage part de 3 037, mais ne converge pas. À la 24^{ième} itération, elle a atteint la valeur de 937.

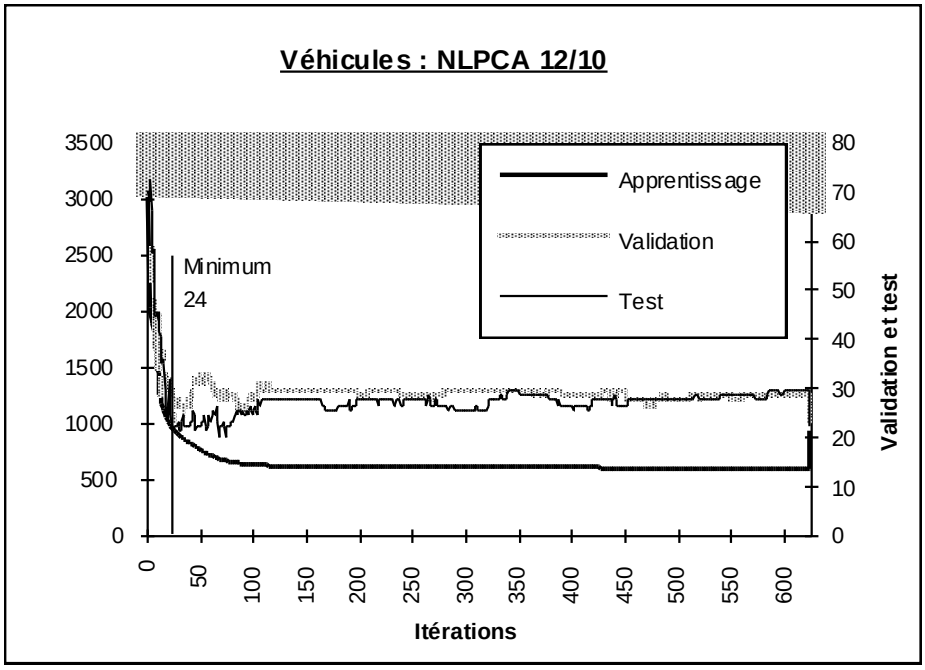


Figure 5.4

3 heures et 53 minutes ont été nécessaires pour réaliser cet apprentissage de 623 itérations. En temps CPU, la machine a travaillé durant 1 heure et 56 minutes. Cependant, après 150 itérations, nous aurions pu conclure avoir les meilleurs résultats et arrêter la simulation. L'apprentissage aurait alors duré moins d'une demi heure en temps CPU.

Les tables 5.1 et 5.2 reprennent les résultats détaillés de la classification réalisée par ce réseau. On constate que ce sont les classes une et deux qui détériorent les résultats. Les autres sont très bien reconnues. Étant donné les mauvaises conditions d'apprentissage, nous pouvons considérer que le réseau a atteint correctement son objectif. La compression NLPCA est réussie.

Véhicules - NLPCA 12/10 - Ensemble d'apprentissage							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	88	61	9	10	0	168	52.3
2	36	113	11	8	0	168	67.2
3	2	3	162	1	0	168	96.4
4	1	1	3	149	0	154	96.7

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	78.2%	74.8%	53.4%
Indécidable :	0%	7.4%	42.2%
Incorrecte :	21.8%	17.8%	4.4%

Table 5.1

Véhicules - NLPCA 12/10 - Ensemble de test							
Classe désirée	Classe obtenue					∑ /classe	% correct
	1	2	3	4	?		
1	14	7	1	2	0	24	58.3
2	5	14	3	2	0	24	58.3
3	1	1	22	0	0	24	91.6
4	1	1	1	21	0	24	87.5

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	74%	71.4%	48.5%
Indécidable :	0%	7.5%	43.6%
Incorrecte :	26%	21.1%	7.9%

Table 5.2

6.3. Compression à 8

Les résultats d'une réduction à 10 étant acceptables, nous avons examiné la compression à 8. Le réseau combiné NLPCA créé pour cela, comprenait cinq couches de 18, 12, 8, 12 et 18 neurones. Nos 658 exemples doivent donc déterminer un réseau de 624 synapses. La règle des dix pourcents n'est toujours pas respectée.

La figure 5.5 décrit le premier apprentissage. La fonction d'apprentissage converge vers 27. Lors de l'arrêt de la simulation, les taux d'erreurs de la fonction test et de la fonction de validation sont respectivement de 16% et 27%.

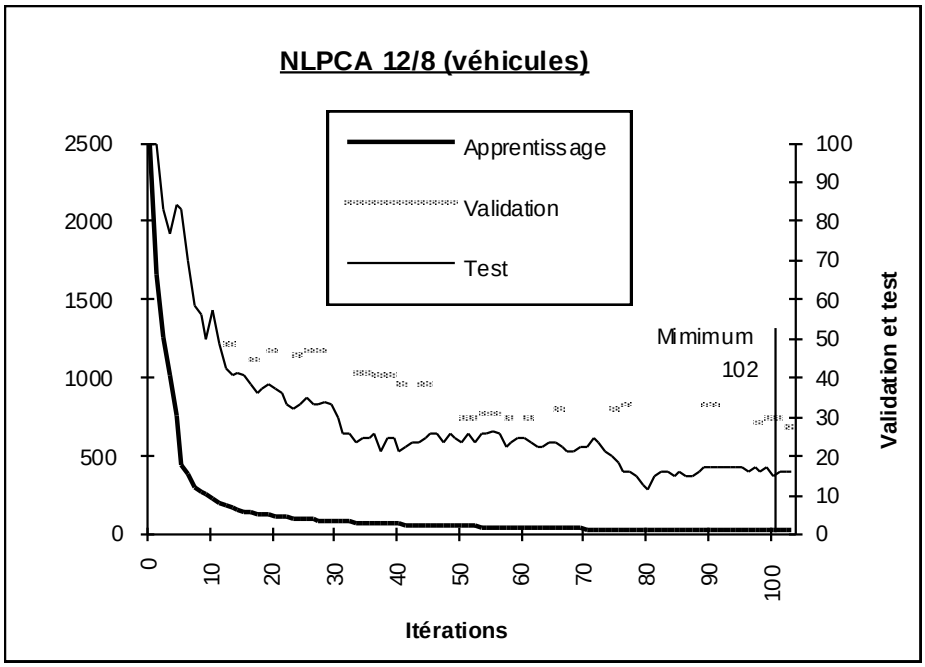


Figure 5.5

1 heure et 59 minutes (1 heure et 55 minutes en temps CPU) ont été nécessaires pour obtenir ces résultats.

L'ensemble comprimé a été récupéré comme expliqué précédemment et fourni au réseau de classification. Ce dernier était constitué de 10 neurones en entrée, 6 en couche cachée et 4 en sortie; cela correspond à 84 synapses. Il y a presque assez d'exemples pour le second apprentissage.

La figure 5.6 montre que son étude a été plus tumultueuse. Le meilleur réseau a été obtenu à l'itération 360, lorsque la fonction de validation indiquait un taux minimum d'erreurs de 20%. Le taux associé de la fonction test est de 25% La fonction d'apprentissage a eu le temps de décroître de 2 612 à 750. La non convergence et l'arrêt sont déclarés à l'itération 491.

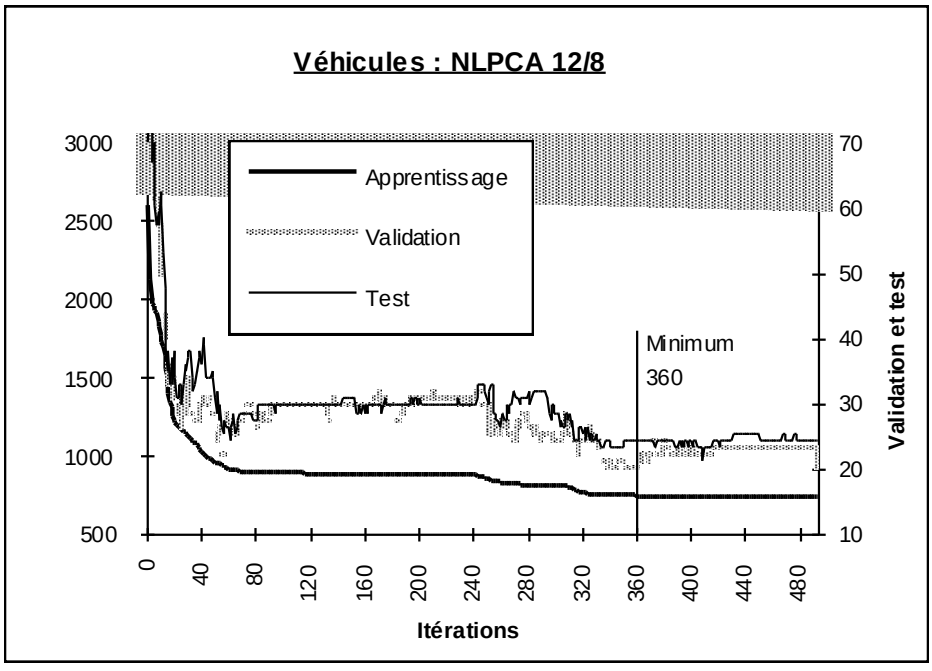


Figure 5.6

Les résultats détaillés des tables 5.3 et 5.4 ont été obtenus après 1 heure et 15 minutes de simulation. Ces tables indiquent que la classe un est mal reconnue, alors que les autres le sont facilement. Les pourcentages globaux indiquant l'exactitude de la classification sont inférieurs à ceux d'une compression NLPCA moins importante, mais excellents relativement aux autres méthodes de réduction : Karhunen et LPC.

Véhicules - NLPCA 12/8 - Ensemble d'apprentissage							
Classe	Classe obtenue					∑	%
désirée	1	2	3	4	?	/classe	correct
1	67	92	3	6	0	168	39.8
2	27	133	5	3	0	168	79.1
3	1	1	165	1	0	168	98.2
4	1	4	1	148	0	154	96.1

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	78.3%	63.9%	56.3%
Indécidable :	0%	26.7%	41.6%
Incorrecte :	21.7%	9.4%	2.1%

Table 5.3

Notons aussi l'utilité d'un critère comme la distance. Ici, les résultats ne sont pas nets. Pour la perfection, il faudrait que les sorties obtenues soient toutes à 0 excepté une seule à 1. Le critère de la distance indique qu'il y a environ 27% des patterns dont les deux plus grandes sorties sont différentes de moins de deux dixièmes, au lieu de 1. C'était une grande source d'erreurs. En appliquant ce critère de sélection, le taux d'erreurs pour l'apprentissage est tombé de 12.9% et pour le test de 18%. Une autre procédure de décision peut être appliquée à ces patterns écartés.

Véhicules - NLPCA 12/8 - Ensemble de test							
Classe	Classe obtenue					∑	%
désirée	1	2	3	4	?	/classe	correct
1	9	14	0	1	0	24	37.5
2	4	18	1	1	0	24	75.0
3	0	1	23	0	0	24	95.8
4	1	1	0	22	0	24	91.6

Classification	Libre	Distance 0.2	Différence 0.5
Correcte :	75%	64.1%	54.2%
Indécidable :	0%	28.7%	42.6%
Incorrecte :	25%	7.2%	3.2%

Table 5.4

6.4. Autres compressions

Nous indiquons maintenant les résultats globaux obtenus pour d'autres réseaux NLPCA de compression. Ils ne sont pas suffisamment significatifs pour faire chacun l'objet d'un paragraphe séparé. La table 5.5 résume nos études. Deux lignes sont associées à chaque tentative : la première décrit l'apprentissage du réseau combiné et la seconde celui du réseau de classification.

Réseau	**Itération**	**Exactitude apprentis.**	**Exactitude test**	**Nombre itérations**	**Temps réel**	**Temps CPU**
18/10/8/10/18	124	2 499 -> 27	88.3%	125	2h 15'	2h 03'
8/6/4	38	73.5%	72.9%	472	2h 37'	1h 14'
18/10/4/10/18	2 144	2 189 -> 59	74.5%	3 000	57h 24'	30h 49'
4/4/4	122	60.2%	64.5%	236	25'	25'
18/8/4/8/18	1 947	1 846 -> 81	69.2%	2 309	39h 18'	21h 37'
4/4/4	300	60.7%	60.2%	375	39'	39'
18/6/4/6/18	650	1 738 -> 96	66.0%	1 418	13h 28'	11h 20'
4/4/4	169	57.6%	49.6%	229	26'	26'

Table 5.5

7. Données corrélées

7.1. Présentation

Le problème de la reconnaissance des véhicules a montré les possibilités de la méthode NLPCA. Cependant, comme cet ensemble de données n'était pas suffisamment important, les résultats ne sont pas exceptionnels. C'est pourquoi nous présentons maintenant l'utilisation de cette méthode sur un ensemble de données créé de toutes pièces de manière à pouvoir être traité efficacement.

Ce problème est un cas d'école très simple. Des vecteurs de deux variables y_1 et y_2 ont été construits. Ils ont été obtenus par la règle suivante :

$$\begin{cases} y_1 = 0.9\sin(x) \\ y_2 = 0.9\cos(x) \end{cases} \qquad x \in [0, 2\pi]$$

Pour l'ensemble d'apprentissage, 400 valeurs de x ont été choisies aléatoirement. L'ensemble de validation comprend 40 vecteurs et l'ensemble de test 400.

Notre objectif est de montrer qu'un réseau NLPCA est capable de retenir parfaitement ces informations dans un seul neurone.

7.2. Résultats

Quatre réseaux combinés ont été envisagés. Chacun a un nombre différent de neurones en couche de mapping, mais ils utilisent tous les mêmes paramètres descriptifs (cfr. table 2.1). Le nombre d'éléments dans le fichier d'apprentissage a été choisi pour respecter largement la règle des dix pourcents. Le plus gros réseau utilisé comporte 36 synapses; c'est-à-dire moins que 400 / 10. Les simulations ont été réalisées par SIRENE à partir des trois ensembles de données. On constate dans la table 5.6 que le neurone de la couche centrale de compression a chaque fois été capable de résumer l'information des deux entrées. Augmenter le nombre de neurones en couches de mapping n'a pas ici amélioré les résultats.

Réseau	Itération	Exactitude apprentis.	Exactitude test	Nombre itérations	Temps réel	Temps CPU
2-6-1-6-2	78	427 -> 25	98%	80	12'	10'
2-4-1-4-2	73	571 -> 25	98%	73	37'	7'
2-3-1-3-2	73	329 -> 26	98%	73	7'	6'
2-2-1-2-2	39	378 -> 33	100%	64	4'	4'

Table 5.6

8. Conclusions

La méthode NLPCA utilise un réseau de neurones pour trouver les caractéristiques non linéaires des données. Le réseau est d'un type conventionnel à propagation : le perceptron. L'architecture particulière utilisée emploie trois couches cachées, incluant une couche intérieure de compression. Ce réseau effectue un apprentissage où l'entrée doit être reproduite à la sortie. En mettant moins de neurones dans la couche de compression que dans la couche d'entrée, le réseau est obligé de trouver des valeurs représentatives dans la couche centrale pour réaliser sa fonction : reproduire son entrée à la sortie. Le fait d'utiliser des fonctions de transfert sigmoïdales ou linéaires pour les neurones de compression, permet au réseau de tenir compte des corrélations non linéaires dans les entrées.

Pour notre objectif de minimisation du temps d'apprentissage, il est clair que la méthode de compression NLPCA est totalement inefficace. Étant donné la complexité du réseau nécessaire, la phase de compression est susceptible de prendre plus de temps à elle seule que la phase de classification sans compression préalable. Cette méthode ne peut dès lors être appliquée qu'à de petits ensembles de données. De plus, pour ceux-ci, le nombre d'exemples pour l'apprentissage doit être suffisamment élevé. Pour obtenir de bons résultats, il faut de préférence respecter la règle des dix pourcents. Cela n'est pas toujours évident vu le nombre élevé de synapses dans un réseau combiné NLPCA. Si toutes ces conditions sont respectées, alors la compression NLPCA est intéressante. Ses résultats devraient être meilleurs que ceux obtenus par la méthode de Karhunen, car cette dernière pourrait être implémentée selon le même principe, mais en utilisant uniquement des fonctions d'activation linéaires pour les neurones. La méthode NLPCA est une amélioration de la méthode de Karhunen-Loève.

CHAPITRE 6
LA MÉTHODE LSP

1. Introduction

Les coefficients LSP (Line Spectral Pair) ont été créés pour représenter les signaux de la parole. Ils sont constitués de paires de valeurs. Les tailles de compression possibles doivent donc également être paires. Ces paramètres sont une autre présentation des coefficients LPC.

On ne nous avait pas demandé d'analyser cette méthode. Nous en avions entendu parler et avons décidé de l'étudier. Nous ne l'avons cependant pas appliquée aux problèmes décrits dans le chapitre deux. Nous expliquerons pourquoi dans la seconde section.

2. La méthode

Dans le chapitre 4, nous avons présenté les coefficients LP. Les coefficients LSP ne sont qu'une réécriture de ceux-ci. Nous avions défini une représentation de la puissance spectrale d'un signal par :

$$P(f) \approx \frac{a_0}{\left|1 + \sum_{k=1}^{M} a_k z^k\right|^2} \qquad (1)$$

où les a_i sont les M coefficients LP à calculer et $z \equiv e^{2\pi i f/\Delta}$.

Si nous définissons maintenant :

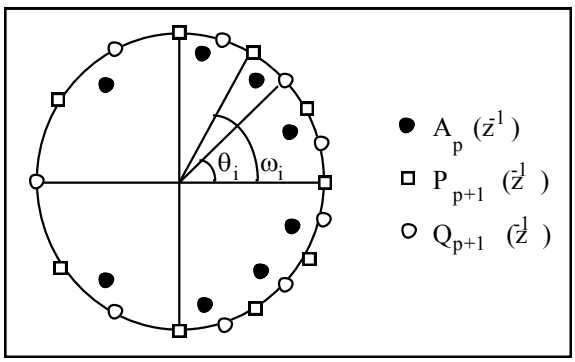
$$A_p(z) = 1 + \sum_{i=1}^{p} a_i z^i \qquad (2)$$

(1) peut se réécrire $1/|A_M(z)|^2$.

Nous pouvons également poser :

$$\begin{aligned} P(z^{-1}) &= A_p\left(z^{-1}\right) - z^{-(p+1)}A_p(z) \\ &= 1 + (a_1 - a_p)z^{-1} + \ldots + (a_p - a_1)z^{-p} - z^{-(p+1)} \\ Q(z^{-1}) &= A_p\left(z^{-1}\right) + z^{-(p+1)}A_p(z) \\ &= 1 + (a_1 + a_p)z^{-1} + \ldots + (a_p + a_1)z^{-p} + z^{-(p+1)} \end{aligned} \qquad (3)$$

c'est-à-dire deux polynômes de degré p+1. Ces polynômes ont une propriété intéressante. Il a été démontré que tous leurs zéros se situent sur le disque unitaire et alternent. De plus, si $e^{i\omega}$ est un de leurs zéros, $e^{-i\omega}$ en est un autre. Remarquons aussi que les polynômes P et Q ont respectivement un zéro en 1 et -1.



En calculant les M coefficients LP, on peut construire les coefficients des polynômes P et Q. Les zéros 1 et -1 ne nous intéressent pas. Quel que soit le signal, ces zéros seront présents et n'apportent donc aucune information supplémentaire. Dans l'implémentation, les polynômes P et Q sont factorisés en deux polynômes P' et Q' d'un degré inférieur et possédant les mêmes zéros à l'exception de -1 et 1. En résolvant P' et Q', on obtient 2M zéros. Cependant, on peut en éliminer la moitié : ceux dont la partie imaginaire est négative. En effet, on a vu qu'il y a toujours un $e^{-i\omega}$ correspondant à $e^{i\omega}$. Ces zéros ne contiennent aucune information supplémentaire. A partir des M coefficients LP, on a donc obtenu M zéros. Étant donné que z = cos (α) + i sin (α), on peut associer à Q, M/2 angles θ_i et à P, M/2 angles ω_i. Par définition, leurs valeurs sont

comprises dans l'intervalle [0,π]. Mathématiquement, en utilisant les paires (ω_i , θ_i), P et Q peuvent être factorisés en :

$$P(z^{-1}) = (1 - z^{-1}) \prod_{i=1}^{p/2} (1 - 2\cos\omega_i z^{-1} + z^{-2})$$

$$Q(z^{-1}) = (1 + z^{-1}) \prod_{i=1}^{p/2} (1 - 2\cos\theta_i z^{-1} + z^{-2})$$

Il faut remarquer que

$$A_p(z^{-1}) = \frac{P(z^{-1}) + Q(z^{-1})}{2}$$

dès lors, les paires (ω_i , θ_i) en sont une représentation et peuvent être utilisées en lieu et place des coefficients LP. On les appelle les coefficients LSP (Line Spectral Pair).

3. Objections à son implémentation

Cette méthode a été imaginée dans le cadre du codage de la parole. Ces coefficients sont préférés aux paramètres LPC, car ils sont en plus étroite relation avec le signal de la parole[2]. Dans ce cas, la valeur de p est généralement 4.

Le premier problème que nous avons rencontré en utilisant cette méthode est la complexité des calculs. Pour chaque pattern de l'ensemble de données à traiter, il faut résoudre deux équations polynomiales dont le degré est égal à la taille de compression. Pour la reconnaissance des phases du sommeil et une taille de compression de 20, nous devions donc résoudre 36 000 équations de degré 20! De plus, il était nécessaire que tous les zéros se trouvent sur le disque unitaire. Malheureusement, que cela soit à l'aide de nos programmes ou de programmes professionnels tels que Mathematica ou Matlab, il y avait souvent un zéro qui ne respectait pas cette condition. Nous n'obtenions donc pas à chaque fois un vecteur réduit entier. Nous supposons que les erreurs proviennent d'imprécisions de calcul. Nous avons essayé de remédier à ce problème avec des spécialistes de Mons, mais sans

[2] N. SUGAMURA, F. ITAKURA, "Speech Analysis and Synthesis Methods Developed at ECL in NTT - from LPC to LSP -",Speech Communication 5, Elsevier Science Publishers B.V., North-Holland, 1986.

succès. Ils n'utilisent pas des valeurs de p aussi grandes. Notons aussi que d'autres méthodes de calcul ont été proposées. Celles qui nous ont été montrées procédaient par approximations successives des coefficients. Cependant, elles nous ont été déconseillées, car les algorithmes d'amélioration de ces paramètres dépendent souvent des types de signaux. Notre objectif étant une méthode générale de compression, nous avons abandonné cette voie.

Outre ce problème de forme, il y en a un de fond. Les coefficients LSP sont avantageux pour représenter des échantillons de parole. Nous ne sommes pas convaincu que les calculs complexes que nécessite cette méthode, se justifie dans notre cas. En effet, les coefficients LSP sont obtenus par des manipulations des coefficients LPC. À priori, un réseau de neurones ne devrait pas être très sensible à ces modifications et devrait même être capable de les représenter. De plus, à un coefficient LPC correspond une paire de coefficients LSP. Pour représenter la même quantité d'informations, il faudra deux fois plus de paramètres. Le gain apporté par les coefficients LSP, s'il existe, ne devrait pas être important. Rappelons aussi que les paramètres LPC donnent déjà d'excellents résultats et qu'il n'est pas évident de pouvoir les améliorer.

4. Conclusions

Nous avons présenté le calcul des coefficients LSP. Différentes méthodes sont possibles pour les obtenir. Celle que nous avons étudiée transforme les coefficients LPC pour obtenir les LSP. Ces derniers se présentent sous forme de paires d'angles compris entre 0 et π.

Nous nous sommes rendus compte de la difficulté de calculer ces paramètres. Cela nécessite énormément de calculs complexes. De plus, le gain apporté par leur utilisation dans un réseau de neurones à la place des coefficients LPC n'est pas évident. Nous sommes convaincus du contraire. Pour ces raisons, nous n'avons pas intégré notre implémentation dans notre programme.

CHAPITRE 7
COMPARAISONS DES MÉTHODES

1. Introduction

Dans ce chapitre, nous allons rassembler nos principaux résultats et les comparer. Notre but est de déterminer quand et comment utiliser chaque méthode, pour quelles raisons et pour quels résultats.

2. Les résultats

La table 7.1 reprend les principaux résultats obtenus dans notre travail. Ils sont classés par ensembles de données et par ordre décroissant de pourcentage de classification exacte de l'ensemble test. Les renseignements qui sont fournis sont : la méthode de prétraitement, le nombre minimum d'itérations pour obtenir le meilleur réseau, pour celui-ci le pourcentage d'exactitude sur l'ensemble d'apprentissage et sur l'ensemble de test, le nombre d'itérations avant l'arrêt de la simulation, le temps réel et CPU qu'elle a nécessité, ainsi que le temps CPU estimatif qui a été nécessaire pour obtenir la meilleure itération.

Pour analyser ces résultats, nous devons d'abord définir nos critères de réussite. Ce qui nous importe le plus est que pour un fichier de données quelconque prétraité, puis présenté au réseau de neurones déjà entraîné, le pourcentage de classification correcte soit le plus élevé possible. Nous avons donc classé les méthodes selon leurs résultats pour l'ensemble de test. Cependant, ce n'est pas le seul facteur à étudier. Notre objectif dans ce travail est de minimiser le temps d'apprentissage des réseaux. Comme on l'a vu, un réseau très petit, c'est-à-dire celui pour lequel on a réalisé une compression importante des données, n'est pas forcément plus rapide qu'un réseau plus complexe. La rapidité tient compte de deux facteurs : le temps pour une itération et le nombre d'itérations nécessaires. La compression n'influence directement que le temps nécessaire à la réalisation d'une itération. L'autre facteur ne peut être deviné. Pour une compression à 10 par la méthode LPC sur les données du sommeil, le temps d'apprentissage du réseau est beaucoup plus long que pour une compression à 20, car il faut 992 itérations au lieu de 141 pour y parvenir. Cela bien que le temps CPU pour

une itération soit passé de 2 minutes 47 secondes à 1 minute. A l'inverse, toujours pour les données du sommeil, la méthode de Karhunen nécessite moins d'itérations pour une compression plus forte. On a donc gagné en temps par itération (2 minutes à 1 minute) et en nombre d'itérations (651 à 486). On peut donc conclure qu'on ne doit pas forcément essayer d'obtenir la compression la plus forte. Lors de l'application de la méthode LPC sur les données du sommeil, les résultats étaient déjà excellents en qualité et en temps pour une compression à 20. Il paraissait déjà difficile d'obtenir de meilleurs résultats selon ces deux critères. La compression à 10, fourni une qualité de classification semblable, mais en un temps beaucoup plus important. Il n'était pas nécessaire de l'envisager.

Méthode	Min.	Exactitude apprentis.	Exactitude test	Max.	Temps réel	Temps CPU	T. CPU estim.
Phases du sommeil							
Sans	1 405	99.7%	98.3%	2 000	484h 25'	442h 48'	311h 04'
LPC 20	141	99.7%	98.1%	141	23h 13'	5h 49'	5h 49'
LPC 10	992	99.6%	97.4%	992	52h 44'	17h 30'	17h 30'
Karhunen 20	651	91.2%	89.5%	1 550	109h 31'	54h 04'	22h 42'
Karhunen 10	486	70.8%	72.9%	492	30h 23'	8h 10'	8h 04'
NLPCA	>	-	-	>	>	>	>
Caractères							
Sans	>	-	-	>	>	>	>
Backpropag.[1]	-	95.9%	95.1%	-			36h 00'
Karhunen 40r	53	96.3%	93.0%	580	87h 20'	86h 31'	7h 54'
Karhunen 20r	1 097	95.3%	91.3%	1 498	291h 42'	285h 44'	209h 14'
LPC 40	1 184	77.0%	74.9%	1 280	263h 12'	170h 31'	157h 43'
NLPCA	>	-	-	>	>	>	>
Véhicules							
Sans	59	88.0%	83.3%	444	1h 29'	1h 22'	10'
Backpropag.[2]	-	83.2%	79.3%	-			4h 00'
Karhunen 10	61	85.7%	76.0%	975	3h 09'	2h 57'	11'
NLPCA 8	462	78.3%	75.0%	598	3h 24'	3h 10'	2h 26'
NLPCA 10	109	78.2%	74.0%	623	7h 29'	3h 38'	38'
LPC 8	62	57.3%	54.1%	610	3h 03'	1h 31'	9'
LPC 10	87	54.6%	44.7%	331	1h 49'	1h 48'	28'
Karhunen 8	4	48.2%	40.6%	404	1h 07'	1h 00'	1'

1. Il s'agit des résultats du projet ESPRIT StatLog obtenus par un perceptron sur des données ramenées à une taille de 40 par la méthode de Karhunen.
2. Ce sont les résultats du projet StatLog sans prétraitement.

> Les résultats sont inconnus, mais sont supposés supérieurs à ceux de la même catégorie.

\- Les résultats sont inconnus.

Table 7.1.

Nos ensembles de données ont illustré parfaitement ce que nous annonçait la théorie. Il est important de remarquer l'ordre des méthodes pour chaque type de données. Pour des signaux comme ceux correspondant aux phases du sommeil, la méthode LPC est en tête. La méthode de Karhunen-Loève suit avec de bons résultats. Pour une même taille de compression, la méthode LPC surclasse largement la méthode de Karhunen; cela d'autant plus que la compression est forte. Même pour une compression supérieure (10) par LPC, la méthode de Karhunen (20) ne peut se défendre, que cela soit en qualité ou en temps nécessaire. Par contre, si nous ne travaillons pas avec des signaux, les performances du LPC sont nettement inférieures. Dans le cas des caractères par exemple, Karhunen fournit toujours de bons résultats même pour une compression importante. La réduction LPC est déjà défavorable pour une taille de 40, la plus mauvaise après la méthode NLPCA. Parlons de cette dernière. Elle ne peut être appliquée qu'à de petits ensembles de données. Pour les autres, il est certain qu'elle ne peut concurrencer LPC ou Karhunen. Par contre, pour un ensemble de taille réduite comme celui des véhicules, elle se défend. Si toutes les conditions d'apprentissage avaient été remplies, ses résultats auraient été meilleurs. Un indicateur de ce fait est qu'une compression à 8 par NLPCA donne de meilleurs résultats qu'une compression moins importante à 10. On sent l'influence du manque d'exemples lors de l'apprentissage. Dans de meilleures conditions d'utilisation, la méthode de Karhunen n'aurait probablement pas gardé la première place. Il faut rappelé que NLPCA est une amélioration par réseau de neurones de la méthode de Karhunen-Loève. Cependant, nous ne pouvons pas modifier le contexte et nous nous contenterons de ces résultats. Pour une taille de compression importante, NLPCA montre ses qualités. Le réseau combiné correspondant a pu être optimisé et souffre moins de la carence en exemples. À la taille 8, Karhunen et LPC ne peuvent suivre. En conclusion, la méthode de Karhunen est toujours conseillée. Si les données représentent un signal, LPC fournira en général de meilleurs résultats encore. La méthode NLPCA est très difficile à mettre en oeuvre. Son usage ne devrait être envisagé que pour de petits ensembles de données.

Nous avons comparé nos prétraitements entre eux. Que valent-ils en général ? Dans le cadre des phases du sommeil, la réussite est éclatante. Nous obtenons le même pourcentage de classification correcte que sans prétraitement, mais 53 fois plus vite! Pour les caractères, nous ne pouvons malheureusement pas établir de comparaisons. Cependant, avec 93% de réussite, l'utilisation de la méthode de Karhunen est très satisfaisante. Notons que la qualité de nos résultats est confirmée par celle très proche du projet ESPRIT StatLog. De plus, le temps nécessaire estimatif est raisonnable pour un réseau de neurones. Enfin, pour les véhicules, nos manipulations

nous ont fait perdre de la précision, mais les résultats ne sont pas mauvais. Avec 7.3% de différence et les conditions de simulation rencontrées, on peut également conclure à la réussite de nos prétraitements. Ici évidemment, on ne voit pas très bien ce qu'on y a gagné. Les temps de simulation sont faussés par les conditions de traitement et trop petits pour être significatifs. La compression NLPCA est cependant beaucoup plus lente. Notons également que nous avons obtenu de meilleurs résultats sans prétraitement que le projet StatLog.

Commentaire [5]: pour utiliser correctement la methode, il faut plus d'exemples -> les temps de simulation augmentent et ne sont plus rentables !!!

3. Conclusions

Trois éléments importants ont été dégagés :

- Tout d'abord, selon nos critères d'efficacité et de temps, la compression la plus forte possible n'est pas toujours la meilleure. La vitesse d'apprentissage dépend de deux facteurs principaux : le temps pour une itération et le nombre d'itérations nécessaires. La compression ne modifie directement que le premier.

- La méthode de Karhunen-Loève donne en général de très bons résultats quel que soit le type des données à traiter. La méthode LPC la surclasse pour des signaux. NLPCA ne devrait être envisagé que pour de petits ensembles de données.

- Si le prétraitement est bien choisi, les résultats obtenus sont tout à fait fiables. La perte de qualité est minime. De plus, les temps de simulation sont fortement diminués comme nous le cherchions. Nous pouvons conseiller ces prétraitements.

CONCLUSIONS

Nous avons commencé notre travail par une description des réseaux de neurones. Théoriquement déjà, on découvre ses nombreuses qualités : parallélisme, capacité d'adaptation, mémoire distribuée, capacité de généralisation, facilité de construction … La pratique confirme ces premières observations. Nous avons utilisé des réseaux de neurones artificiels pour résoudre plusieurs problèmes complexes, commerciaux et industriels. Les résultats ont été excellents et des comparaisons avec d'autres méthodes montre que le RNA est tout à fait compétitif.

Cependant, la théorie laisse également deviner un inconvénient du RNA : la complexité de sa structure. Les RNA seront optimums quand ils auront leur propre support et pourront exploiter pleinement le parallélisme. La taille des réseaux rend malheureusement de telles implémentations encore plus difficiles à réaliser. Actuellement, des simulateurs sur ordinateurs sont généralement utilisés. Hélas, pour eux aussi, la complexité des réseaux est un gros problème, d'autant plus que les simulateurs sont habituellement séquentiels. Le nombre de calculs nécessaires à l'apprentissage d'un réseau devient très vite phénoménal. Il apparaît dès lors comme une priorité de trouver des méthodes pour réduire la complexité des RNA.

Il nous a été demandé d'en analyser trois : Karhunen-Loève, LPC et NLPCA. Nous en avons proposé une quatrième : LSP. La taille d'un réseau est liée au nombre de ses entrées. Si on peut le diminuer, la complexité du réseau décroît. Les 4 méthodes que nous avons étudiées, prétraitent les valeurs fournies aux entrées d'un réseau pour obtenir un nouveau pattern de caractéristiques de taille inférieure. Si cette transformation est bien réalisée, un réseau plus petit utilisant ces données sera toujours capable de réaliser son apprentissage et de fournir des résultats semblables à ceux obtenus par le réseau n'utilisant pas le prétraitement. Le but principal de notre travail était d'étudier les qualités de ce prétraitement pour les méthodes demandées.

Elles ont été utilisées pour résoudre différents problèmes de classification. Nous avons été agréablement surpris. Les résultats ont été excellents et surtout n'ont pratiquement pas été dégradés par les prétraitements. De plus, les temps de simulation ont été diminués considérablement. Nous étions déjà convaincus de l'utilité de ces prétraitements et cependant, les résultats ont dépassé nos espérances. Nous ne pouvons que conseiller d'utiliser ces méthodes.

Nous devons cependant les distinguer. Toutes ne sont pas aussi efficaces, ou ne le sont que dans certains cas. La méthode de Karhunen-Loève donne en général de très bons résultats quel que soit le type de données à traiter. La méthode LPC la surclasse cependant pour des données de signaux. NLPCA est très lourd à utiliser et ne devrait être envisagé que pour de petits ensembles de données. Dans de bonnes conditions d'utilisation, elle donnera de meilleurs résultats que la méthode de Karhunen. La compression par LSP a été abandonnée. Elle est trop difficile à réaliser et ne devrait pas apporter de meilleurs résultats que la méthode LPC sur laquelle elle est basée.

Une dernière remarque concerne l'importance des compressions. Le réseau le plus petit n'est pas forcément le meilleur. Dans le cadre de nos simulations, il a été montré qu'il faut trouver un juste milieu. Si la compression est faible, le réseau ensuite utilisé reste imposant et le gain est minime. Si la compression est trop forte, le réseau n'a plus assez d'informations déterminantes et l'apprentissage peut demander un plus grand nombre d'itérations, c'est-à-dire plus de temps. Malheureusement, il n'y a pas de règle permettant de deviner la taille idéale, celle qui allie réduction significative de la complexité du réseau et gain de temps pour son apprentissage. Avec un peu d'expérience, elle est cependant rapidement trouvée.

Ce travail nous a convaincu de l'utilité des réseaux de neurones et des prétraitements. Ces derniers se sont révélés très efficaces et très faciles à réaliser. Il ne faut surtout pas les négliger.

BIBLIOGRAPHIE

- Jacek M. ZURADA, *"Introduction to Artificial Neural Systems"*, West Publishing Company, 1992.
- P. LATOUR, *"Utilisation des réseaux de neurones artificiels dans la reconnaissance de formes en signaux physiologiques"*, Convention FIRST, région wallonne, université de Liège, avril 1990.
- F. BLAYO, "*Réseaux neuronaux"*, laboratoire de Microinformatique, avril 1992.
- W. H. PRESS, B. P. FLANNERY, S. A. TEUKOLSKY, W. T. VETTERLING, *"Numerical Recipes"*, Cambridge University Press.
- E. DAVALO, P. NAÏM, *"Des réseaux de neurones"*, deuxième édition, éditions Eyrolles, 1990.
- F. X. LITT, *"Analyse numérique"*, notes de cours, Faculté des Sciences Appliquées, Université de Liège, 1991.
- J. ETIENNE, *"Analyse mathématique"*, notes de cours, Faculté des Sciences Appliquées, Université de Liège, 1986.
- M. BODESON, *"Reconnaissance de mouvements mandibulaires par neurones artificiels"*, travail de fin d'études pour l'obtention du grade de Licencié en Informatique, 1991-1992.
- P. LASCAUX, R. THEODOR, *"L'analyse en composantes principales"*, Analyse numérique matricielle appliquée à l'art de l'ingénieur, tome 1.
- K. FUKUNAGA, W. L. G. KOONTZ, *"Application of the Karhunen-Loève Expansion to Feature Selection and Ordering"*, IEEE Transactions, vol. C-19, number 4, April 1970.
- *"The Karhunen-Loève Expansion"*, Spectral Analysis.
- N. SUGAMURA, F. ITAKURA, *"Speech Analysis and Synthesis Methods Developed at ECL in NTT - from LPC to LSP -"*,Speech Communication 5, Elsevier Science Publishers B.V., North-Holland, 1986.
- J. HANCQ, *"Automatic scoring of sleep stages with LSP adaptive filtering"*, IEEE Benelux & ProRISC, Proceedings of the Workshop on Circuits, Systems and Signal Processing, Houthalen, April 1992.
- F. W. ZAKI, *"Learning Characteristics of a New Adaptive Line Spectral Pair Filter",* Mu'tah University, Jordan, Submitted to publication (Signal Processing).
- M. A. KRAMER, *"Nonlinear Principal Component Analysis Using Autoassociative Neural Networks"*, AIChe Journal, Vol. 37, number 2, February 1991.

- E. SAUND, *"Dimensionality-Reduction Using Connectionist Networks"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, number 3, March 1989.
- J. MAKHOUL, *"Linear Prediction : A Tutorial Review"*, Proceedings of the IEEE, Vol. 63, number 4, April 1975.
- R. D. KING, R. C. HENERY, A. SUTHERLAND, *"A comparative Study of Classification Algorithms : Statistical, Machine Learning, and Neural Network (Draft)"*, August 1992.

Annexe A
Menus de SIRENE

```
*********************************************
*                                           *
*                                           *
*              SIRENE V1R9                  *
*        SImulateur de REseaux de NEurones  *
*                                           *
*              M. Fombellida                *
*              Service de microelectronique *
*              Institut Montefiore          *
*              ULG                          *
*                        October 92         *
*********************************************

          ******************
          *   MAIN  MENU   *
          ******************


 1:Network...
 2:Patterns...
 3:Algorithms...
 4:Learn
 5:Use
 6:Statistics...

99:QUIT

Your choice :
```

```
       ********************
       *   NETWORK MENU   *
       ********************


 1: New network...
 2: Edit current network...

 3: Load BIN network file
 4: Save network in BIN file
```

```
 5: Load ASCII network file
 6: Save network in ASCII file

 7: Convert BIN network file in ASCII file
 8: Convert ASCII network file in BIN file

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
        *************************
        *   NEW NETWORK MENU  *
        *************************


 1: Create a multiple layers perceptron with full connexion
 2: Create a input-output perceptron with full connexion

 0: RETURN TO MAIN MENU
99: QUIT

Your choice : 1


Number of layers:
Number of neuron(s) in layer 1:
Number of neuron(s) in layer 2:

Select the activation functions of the neuron
 1: Linear
 2: Sigmoid
 3: Sine
 4: Cosine
 5: Gaussian
 6: Cosine*Gaussian+Sigmoid
 7: Sigmoid Prime
 8: Hyperbolic tangent
 9: Sigmoid symetric
Your choice : f=tanh(a*x)
a:

...

Number of neuron(s) in layer 3:


Select the activation functions of the neuron
```

```
 1: Linear
 2: Sigmoid
 3: Sine
 4: Cosine
 5: Gaussian
 6: Cosine*Gaussian+Sigmoid
 7: Sigmoid Prime
 8: Hyperbolic tangent
 9: Sigmoid symetric
Your choice : f=tanh(a*x)
a:

...
```

```
      **********************
      *  PATTERNS MENU   *
      **********************



 1: Load patterns file...
 2: Convert patterns file...
 3: Create patterns file

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
      ***************************
      *  LOAD PATTERNS MENU   *
      ***************************



 1: Load BIN file for learning
 2: Load BIN file for validation
 3: Load BIN file for test

 4: Load ASCII file for learning
 5: Load ASCII file for validation
 6: Load ASCII file for test

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
*****************************
*  CONVERT PATTERNS MENU   *
*****************************


 1: Simple file type conversions...
 2: Preprocessings...
 3: Analysis of patterns file

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
*****************************************
*  SIMPLE FILE TYPE CONVERSIONS MENU  *
*****************************************


 1: Convert BIN file in ASCII file
 2: Convert ASCII file in BIN file

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
*************************
*  PREPROCESSING  MENU  *
*************************


All these preprocessings need the analysis file

 1: Convert BIN file in BIN mean-centered (0,1) file
 2: Convert BIN file in BIN mean-centered (-0.5,+0.5) file
 3: Convert BIN file in BIN mean-centered (-1,+1) file
 4: Convert BIN file in BIN mean-centered input (-1,+1) file
 5: Convert BIN file in BIN normalized (0,1) file
 6: Convert BIN file in BIN normalized (-0.5,+0.5) file
 7: Convert BIN file in BIN normalized (-1,+1) file
```

```
 8: Convert BIN file in BIN input-decorrelated file
 9: Convert BIN file in BIN input-divide-by-max file
 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
      **********************
      *  ALGORITHMS MENU *
      **********************


 1: Learning cost functions
 2: Validation cost functions
 3: Test cost functions
 4: Success criteria
 5: Optimization algorithms
 6: Parameters

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
      *******************************
      *   LEARNING COST  FUNCTIONS   *
      *******************************


Select the function to minimize during learning
 1: Total sum squared error
 2: Total sum squared error + minimize weights (W^2)
 3: Total sum squared error + minimize weights (W^2/1+W^2)

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :
```

```
      ***********************************
      *   VALIDATION COST  FUNCTIONS    *
      ***********************************
```

Select the validation cost function
 1: Total sum squared error
 2: Classification error (%) (Maximum criteria)
 3: Classification error (%) (Threshold with margin criteria)

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :

```
**************************
*  TEST COST  FUNCTIONS   *
**************************
```

Select the test cost function
 1: Total sum squared error
 2: Classification error (%) (Maximum criteria)
 3: Classification error (%) (Threshold with margin criteria)

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :

```
**********************
* SUCCESS CRITERIA  *
**********************
```

Select the success criteria of the learning algorithm

 1: Small individual error
 2: Small composite error
 3: Sharp threshold
 4: Threshold with margin

 0: RETURN TO MAIN MENU
99: QUIT

Your choice :

```
**************************************
*   OPTIMIZATION ALGORITHMS MENU  *
**************************************
```

Select the learning algorithm

Heuristic methods
1: Back Propagation with momentum
2: Silva-Almeida
3: Extended Delta-Bar-Delta
4: Extended Quickprop

Non constraint optimization methods
5: Gradient + line search (Steepest descent)

6: Conjugate Gradient (Fletcher-Reeves) + line search
7: Conjugate Gradient (Polak-Ribiere) + line search
8: Limited memory quasi-Newton + line search
9: Quasi-Newton (DFP) + line search
10: Quasi-Newton (BFGS) + line search

11: Conjugate Gradient (Fletcher-Reeves) with restart + line search
12: Conjugate Gradient (Polak-Ribiere) with restart + line search
13: Limited memory quasi-Newton with restart + line search
14: Quasi-Newton (DFP) with restart + line search
15: Quasi-Newton (BFGS) with restart + line search

0: RETURN TO MAIN MENU
99: QUIT

Your choice :

Current value of the parameters

Success criteria: Individual error = 0.002500
Threshold = 0.000000
Up margin = 0.100000
Down margin = 0.100000
Epoch = 1000
Minimum progress = 0.000001
Objective function: Gamma = 0.001000

```
Optimization: Learning rate =        1.500000
        Momentum =                   0.900000
        Mu =                         1.000000
        Kappa =                      0.010000
        Kappam =                     0.100000
        Phi =                        0.100000
        Phim =                       0.500000
        gl =                         20.000000
        gm =                         5.000000
        Theta =                      0.700000
        Up =                         1.200000
        Down =                       0.600000
Statistics: Size =                   100
Randomize: Weights range =           0.500000

FLAGS

Learning =          1

Pruning =           0
-> Relearning =     0

Incremental learning =  0
-> Clamping =       0
  -> Relearning =   0

Incremental pruning  =  0
-> Relearning =     0

Overlearning detection = 0
->Backtracking =    0

Brainwashing =      0

Press RETURN to continue
Modify:
 1:Individual error
 2:Threshold
 3:Up margin
 4:Down margin
 5:Epoch
 6:Minimum progress

 7:Gamma

 8:Learning rate
 9:Momentum
10:Mu
11:Kappa
12:Kappam
13:Phi
```

14:Phim
15:gl
16:gm
17:Theta
18:Up
19:Down
20:Statistic size
21:Weights range

22:Toggle learning flag

23:Toggle pruning flag
24:Toggle -> relearning flag

25:Toggle incremental learning flag
26:Toggle -> clamping flag
27:Toggle -> relearning flag

28:Toggle incremental pruning flag
29:Toggle -> relearning flag

30:Toggle overlearning detection flag
31:Toggle -> backtracking flag

32:Toggle brainwashing flag

0:RETURN TO MAIN MENU
99:QUIT

Your choice :

ANNEXE B
LES MENUS DU PROGRAMME

+++ PRETRAITEMENT DE DONNEES EN RECONNAISSANCE DE FORMES PAR RNA +++

0. Aide

1. Karhunen
2. LPC
3. NLPCA
4. Outils d'aide

9. Quitter

Choix :

+++ KARHUNEN - LOEVE +++

0. Aide

1. Vecteurs propres
2. Matrice de transformation
3. Creation fichier compresse

9. Retour au menu principal

Choix :

+++ VECTEURS PROPRES +++

0. Aide

1. Calcul d'une nouvelle matrice de vecteurs propres
2. Chargement d'une ancienne matrice
3. Sauvegarde d'une nouvelle matrice

9. Retour au menu precedent

Choix :

+++ CALCUL DES VECTEURS PROPRES +++

Nom du fichier d'apprentissage : +++ Initialisation +++
+++ Creation de la matrice d'autocorrelation +++
+++ Recherche des vecteurs propres +++
Nombre de patterns traites :
Taille d'un pattern :

--- ENTER ---

+++ MATRICE DE TRANSFORMATION +++

0. Aide

1. Calcul d'une nouvelle matrice de transformation
2. Chargement d'une ancienne matrice
3. Sauvegarde d'une nouvelle matrice

9. Retour au menu precedent

Choix :

+++ CALCUL D'UNE MATRICE DE TRANSFORMATION +++

Choix du nombre de sorties :
1. Constant
2. Selon importance des valeurs propres
3. Selon importance des valeurs propres avec nombre max

Les valeurs propres sont comprises entre |...| et |...|

Choix :
Nombre de sorties :
Nombre de sorties souhaitees :

--- ENTER ---

+++ COMPRESSION +++

0. Aide

1. Sans estimation de la perte d'infos
2. Avec estimation de la perte d'infos

9. Retour au menu precedent

Choix :

+++ CALCUL DES ERREURS +++

0. Aide

1. Erreur pour chaque pattern (fichier)
2. Erreur moyenne globale (ecran)

9. Retour au menu precedent

Choix : +++ COMPRESSION +++

Nom du fichier a traiter :
Nom du fichier resultat :
Erreur moyenne globale :

+++ LINEAR PREDICTION CODE +++

0. Aide

1. Compression

9. Retour au menu principal

Choix :

+++ OUTILS POUR LA METHODE DE COMPRESSION NLPCA +++

0. Aide

1. Conversion des donnees au format NLPCA
2. Selection des premieres couches d'un reseau
3. Modification du nombre de sorties d'un fichier de donnees
4. Utilisation type d'un reseau de SIRENE
5. Recuperation des sorties d'un reseau

9. Retour au menu principal

Choix :

+++ OUTILS POUR L'UILISATION DE SIRENE+++

0. Aide

1. Creation d'un fichier type d'instructions
2. Modification du nombre de sorties d'un fichier de donnees
3. Utilisation type d'un reseau de SIRENE
4. Analyse des resultats

9. Retour au menu principal

Choix :

+ CREATION D'UN FICHIER D'INSTRUCTIONS NLPCA TYPE POUR SIRENE +

Le fichier d'instructions est sauve sous le nom 'sirene.instr' dans le repertoire courant
Attention, un ancien fichier de ce nom sera remplace !!
Voulez-vous continuer (o/n) ?

Nombre de couches : 3
Nombre de neurones dans la couche 1 :
Nombre de neurones dans la couche 2 :
Nombre de neurones dans la couche 3 :

Nom du fichier d'apprentissage :
Nom du fichier de validation :
Nom du fichier de test :

La fonction par defaut des neurones est la 8 (th) avec 1 comme parametre
L'algorithme d'optimisation choisi est le 10 (Quasi-Newton(BFGS) + line search)
La fonction d'apprentissage est la 2 (Total sum squared error + minimize weights (W^2))
La fonction de validation est la 2 (Classification error () (Maximum criteria))
La fonction de test est la 2 (Classification error () (Maximum criteria))
Creation terminee

--- ENTER ---

A bientot j'espere